TECHNISCHE UNIVERSITÄT
CHEMNITZ

Evaluation of Lane Detection Algorithms based on an Embedded Platform

Master Thesis

Submitted in Fulfilment of the
Requirements for the Academic Degree
M.Sc.

Dept. of Computer Science
Chair of Computer Engineering

Submitted by: Trung Bao Nguyen
Student ID: 362252
Date: 13.06.2017

Supervisors: Prof. Dr. Wolfram Hardt
M.Sc. Felix Hänchen
Dipl. Stephan Blokzyl

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Aufgabenstellung

zur

Abschlussarbeit
im Studiengang Master Automotive Software Engineering

für

Herrn Trung Nguyen Bao
geb. am 21. Juli 1989 in Phu Tho

zum Thema

Evaluation of Lane Detection Algorithms based on an Embedded Platform

Betreuer/ Prüfer:	Prof. Dr. Wolfram Hardt
Ausgabedatum:	24.01.2017
Abgabedatum:	03.07.2017
Tag der Abgabe:	

Unterschrift: _____

Prof. Dr. F. Hamker
Vorsitzender des Prüfungsausschusses

Abstract

Real-time lane detection or localization is a crucial problem in modern Advanced Driver Assistance Systems (ADAS), especially in Automated Driving System. This thesis estimates the possibility to implement a lane detection system in the available low-power embedded hardware. Various state-of-the-art Lane Detection algorithms are assessed based on a number of proposed criteria. From the result of the evaluation, three different algorithms are constructed and implemented in the hardware using OpenCV library. The lane detection stage is done with different methods: using Hough Transform for line detection or randomly sampling hypotheses which are straight lines or cubic splines over the pre-processed binary image. Weights of the hypotheses are then calculated based on their positions in the image. The hypothesis which has highest weight or best position will be chosen to represent lane marking. To increase the performance of the system, tracking stage is introduced with the help of Particle Filter or Kalman Filter. The systems are then tested with several different datasets to evaluate the speed, performance and ability to work in real-time. In addition, the system interfaces with CAN bus using a CAN interface, so that the output data can be sent as messages via the CAN bus to other systems.

Keywords: lane-position detection, lane tracking, image processing, embedded system

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ADAS	Advanced Driver Assistance Systems
CAN	Controller Area Network
FPS	Frames Per Second
GPS	Global Positioning System
IDE	Integrated Development Environment
IPM	Inverse Perspective Mapping
LIDAR	LIght Detection And Ranging
OpenCV	Open Source Computer Vision Library
RADAR	RAdio Detection And Ranging
RANSAC	RANdom SAmple Consensus
ROI	Region Of Interest

1 Introduction

1.1 Motivation

Each year, car accidents kill thousands of people around the world. Only in Germany, the number is 3459 people in 2015 and 88% of these accidents are caused by driver faults [27]. This huge number may be reduced significantly by using Advanced Driver Assistance Systems (ADAS). ADAS has been developed to assist the driver and enhance the safety of driving with the help of several different modern technologies. One of them is lane-position detection. For the last few decades, this useful technology has received a considerable amount of attention from researchers in the field around the world.

A lane-position detection system mostly uses image processing techniques to find lane markings from the input video captured by a single camera on the dashboard of the vehicle. There are three main objectives of a lane-position detection algorithm, which are: (i) *Lane Departure Warning System*, (ii) *Driver Attention Monitoring System*, (iii) *Automated Vehicle Control System*. Among them, the last objective is considered to be the most complicated one and has emerged as the top technology trend recently. The common challenges in lane detection can be listed as below:

- The system is mostly required to work in real-time. In fact, recent researches always firstly mention the ability to work in real-time before discussing the accuracy of the system.
- Lane markings are faded or covered by other vehicles. Some examples of this challenge are in Figure 1.1.
- Road surface is not flat. A popular approach in most of researches is using the assumption that the road surface is flat. Until the present moment, there are only a few researches try to deal with the problem of 3D road surface like Leonard et al. [19], because it normally requires input data from different types of sensors.
- Road with high curvature. The curvature of the road is one of the crucial factors that needs to be taken into account in lane detection. Several researches assume that lane markings are straight lines, for example in Kuk et al. [18] and Voisin et al. [29], some use more complicated models such as B-Spline to represent lane markings Wang et al. [30]. Other researches like [5] try to

reduce the curvature of lane markings in the image by using bird's-eye view created by applying a technique called Inverse Perspective Mapping (IPM).

- Other challenging scenarios: high variation of lane width, crossroads, etc.

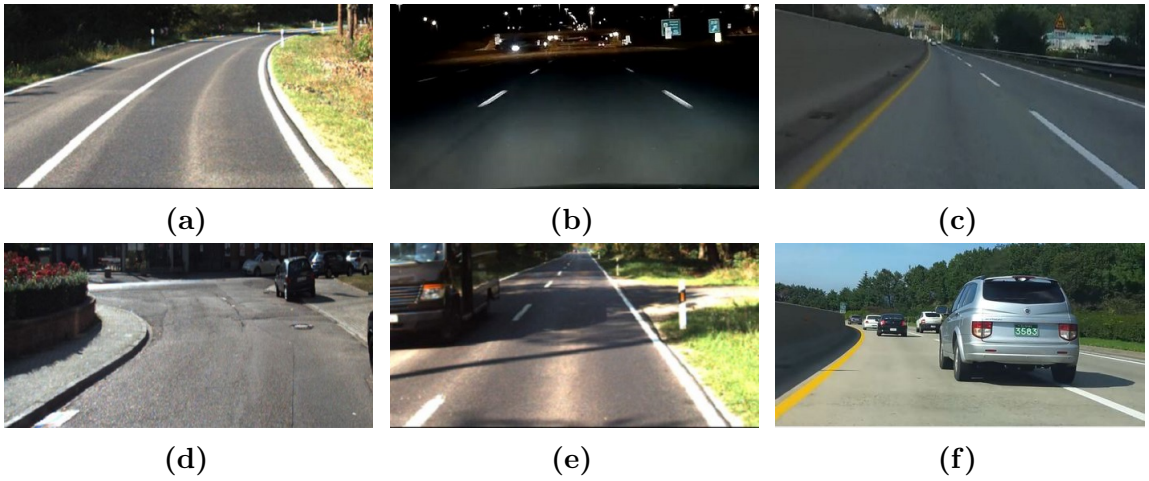


Figure 1.1: Examples of different scenarios in lane-position detection and tracking. a) Road with curving lane markings; b) Lane markings in night condition; c) Road with solid lane marking in yellow color and dash-line lane marking in white color; d) Road with no or faded lane markings; e) Road in critical shadow condition; f) Lane marking is covered by other vehicle.

With these conditions in mind, this thesis aims to:

- Estimate the possibility that a lane detection system may be implemented in our available low-power embedded hardware.
- Implement in the hardware a lane-position detection system that uses input images from a monocular vision camera. The system then needs to be able to work in real-time to find positions of lane markings in the images.

1.2 List of Contributions

Contributions of the thesis to the topic are as follow:

- Evaluate several different state-of-the-art researches in lane-position detection.
- Estimate the possibility of implementing a lane-position detection system in a embedded low-power hardware.
- Develop C++ code for three different lane detection algorithms using the techniques: Inverse Perspective Mapping, Hough Transform, lane tracking with Particle Filter, modeling lane markings with straight line and cubic spline, etc.

- Test the lane-position detection system in several datasets with different road scenarios and varying light conditions.

1.3 Structure of the Thesis

This thesis is organized as following:

- **Chapter 2: Background**
  This chapter describes the background concepts to understand the lane detection and tracking algorithms in the thesis.

- **Chapter 3: State of the Art in Lane Detection**
  In this chapter, we present several different models and approaches from relevant literature of lane detection and tracking. In addition, the advantages and disadvantages of each methods are also discussed.

- **Chapter 4: Concepts**
  Methods of each algorithm and techniques used in the thesis will be explained in detail in this chapter.

- **Chapter 5: Implementation**
  This chapter describes the implementation of each algorithm at length. It also provides comprehensive information of the tools and software used in the thesis for implementing lane detection system.

- **Chapter 6: Testing and Evaluation**
  This chapter provides the information on datasets and testing criteria used in the thesis. Besides, it presents and discusses the results under testing and evaluation of each algorithm.

- **Chapter 7: Conclusion and Future Work**
  This chapter ends the thesis with some concluding remarks and proposes the direction for future development.

2 Background

This chapter describes several general concepts in image processing techniques as well as lane detection and tracking algorithms.

2.1 Computer Vision and Digital Image Processing

Computer Vision

Computer vision is an interdisciplinary field, in which we are trying to *"describe the world that we see in one or more images and to reconstruct its properties, such as shape, illumination, and color distributions"* [12]. In other words, computer vision provides us the understanding of the scene.

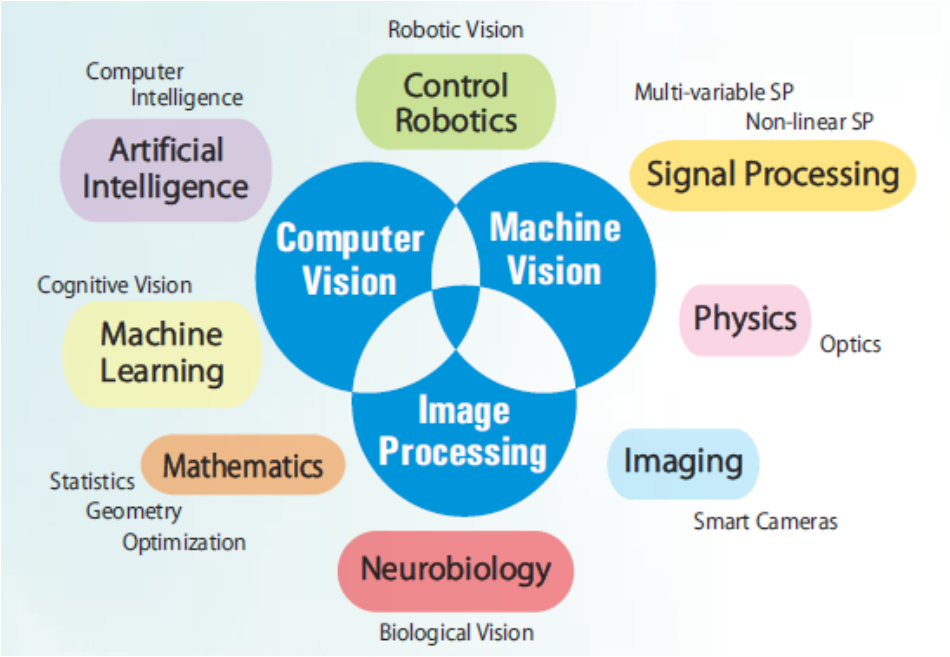


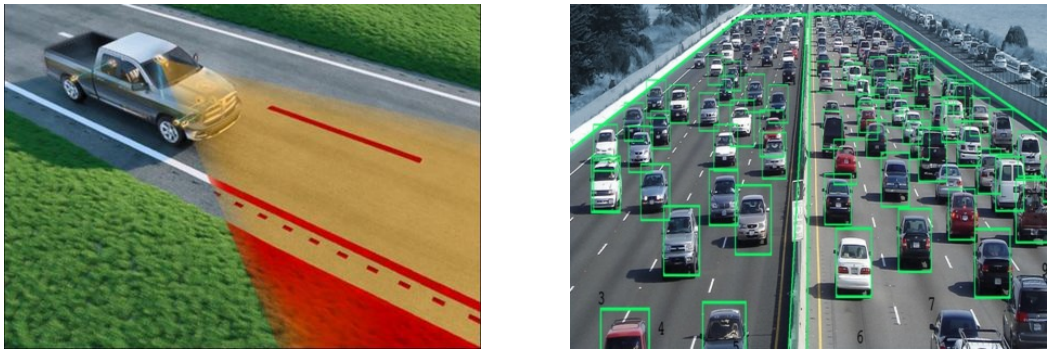
Figure 2.1: Domains of Computer Vision
http://miracleeyegroup.com/imaging-solutions/

In general, machine vision tasks include methods for:

- **Acquiring**: How do the sensors, for example cameras, acquire images of the surrounding world? The way those images encode characteristics of the scene, such as vertices, edges, materials, illumination, etc.?

- **Processing**: Based on how the information about the surrounding world is encoded, the images are processed by applying several techniques of signal processing.
- **Analyzing**: What are the methods or algorithms used to extract meaningful information from the processed images and provides the descriptions of the scene?
- **Understanding**: How are the descriptions of the objects and surrounding world stored and represented?

Nowadays, computer vision is being used widely in various kinds of applications in real life, for instance, optical character recognition (OCR), medical imaging, automotive safety, surveillance, etc.



(a) **(b)**

Figure 2.2: Some applications of computer vision in real life. a) Automotive safety *http://www.continental-automotive.com/*; b) Surveillance and traffic monitoring *http://www.trafficvision.com/*

Digital Image Processing

Image processing is a type of signal processing, which has an image, series of images or video frames as the input, while the output is processed images, maybe in a different format. Image processing is often exploited in computer vision to manipulate digital images.

Vision can be considered the most important part of human perception and it plays the same role in machine sensing. However, unlike human, machines "see" the world as digital images of 2D or 3D scenes produced by sensors. The 2D digital image, which, in many cases, represents a projection of a 3D scene, may be defined as a two-dimensional array of intensity samples called pixels (Figure 2.3). Pixel values typically represent gray levels, colors, opacities, etc. 2D digital image can be processed and manipulated by computer programs to produce useful information about the world.

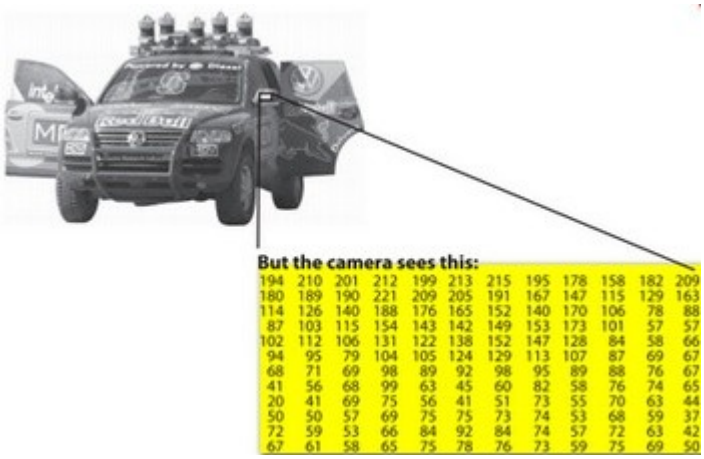


Figure 2.3: Digital image [24]

Several techniques in digital image processing are noise removal, image blur, image sharpening, object recognition, etc.

2.2 OpenCV Library

Most of the image processing techniques in this thesis, for example image reading, grayscaling, thresholding, etc., are realized by using Open Source Computer Vision Library (OpenCV), which is the most popular open-source library in computer vision. From the OpenCV Reference Manual [25]: *"OpenCV (Open Source Computer Vision Library) is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms"*. OpenCV is written in optimized C/C++ language and has a modular structure.

Figure 2.4: OpenCV logo *http://opencv.org/*

Four modules used in the thesis are listed as below:

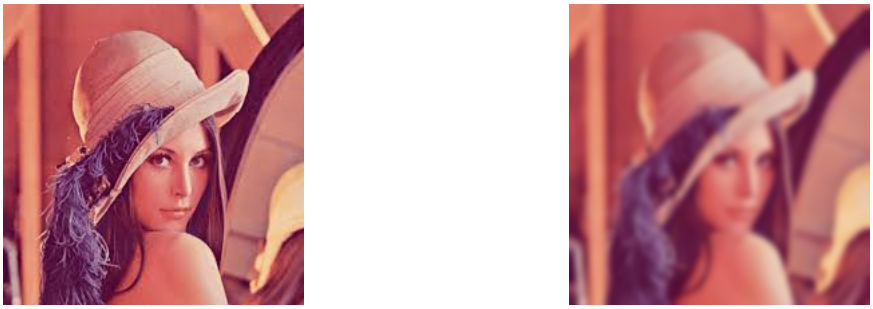
- **core**: this module defines basic data structures and basic functions used by all other modules.

- **imgproc**: this is the image processing module, which includes functions to manipulate images, for example image filtering, resizing, color space conversion, etc.
- **video**: this module provides some tools for analyzing videos such as motion assessment and object tracking.
- **highgui**: this module provides several simple UI capabilities, video capturing and video recording.

Listed below are several examples of using OpenCV in image processing:

- Gaussian Smoothing or Gaussian Blurring: the most commonly used blurring method which blurs an image using a Gaussian filter.

```
void GaussianBlur(InputArray src, OutputArray dst, Size ksize
 , double sigmaX, double sigmaY=0, int borderType=
 BORDER_DEFAULT)
```

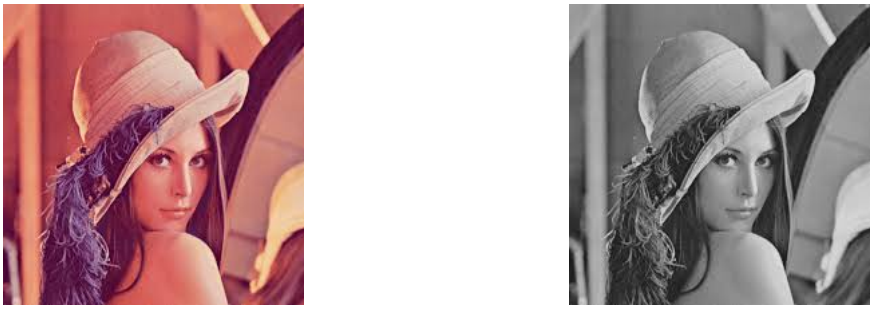


(a) Original image **(b)** Blurred image

Figure 2.5: Gaussian Blur with kernel size 13×13

- Grayscaling: converts an image from RGB color space to grayscale.

```
void cvtColor(InputArray src, OutputArray dst, CV_BGR2GRAY,
 int dstCn = 0)
```



(a) Original image **(b)** Grayscale image

Figure 2.6: Grayscaling

- Thresholding: transforms a grayscale image to a binary image

```
double threshold(InputArray src, OutputArray dst, double
 threshold, double max_value, CV_THRESH_BINARY)
```

(a) Original image

(b) Thresholding image

Figure 2.7: Thresholding with $threshold = 150, max\_value = 255$

2.3 Kalman Filter

Kalman Filter is an extremely famous algorithm in signal processing. It was first introduced in 1960 by Rudolph E. Kálmán and is considered as *"optimal recursive data processing algorithm"* [21] under *"a strong but reasonable set of assumptions"* [8]. Bradski and Kaehler [8] have mentioned three important assumptions in Kalman Filter:

1. The system being modeled must be linear.
2. The noise that measurements are subject to is "white", or in other words, "not correlated in time".
3. The noise is Gaussian in nature.

Under those assumptions, Kalman filter has the ability to estimate in current time step a new model for the state of the system that is most possibly accurate, taking into account the model in the previous time step with its uncertainty and the current measurement with its uncertainty (Figure 2.8).

The state x of the system at time step t is generalized by the following function of the state at time step $t-1$:

$$x_t = Ax_{t-1} + Bu_t + w_t \tag{2.1}$$

with

- x_t and x_{t-1} are n-dimensional state vector at time step t and $t-1$.

- u_t is an c-dimensional control vector at time step t.
- `A` is an $n \times n$ matrix that associates the state at previous time step $t-1$ to the state at current time step t.
- `B` is an $n \times c$ matrix that associates the control input u to the state x.
- w_t is a random variable that represents the noise of state transition. The components of w_t are assumed to have Gaussian distribution $N(0, Q)$, with Q is a $n \times n$ covariance matrix.

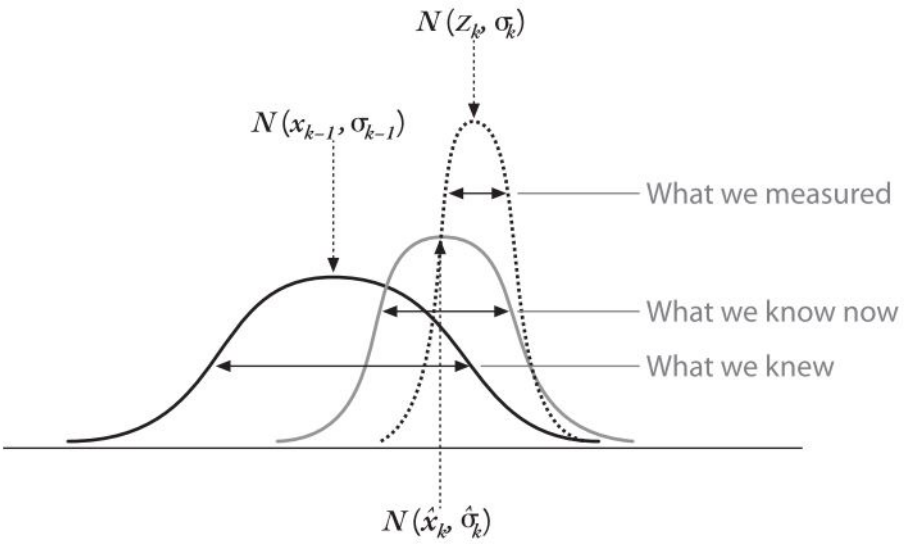


Figure 2.8: The new estimate $N(\hat{x}_t, \hat{\sigma}_t)$ is the combination of prior knowledge $N(x_{t-1}, \sigma_{t-1})$ and measurement observation $N(z_t, \sigma_t)$ [8]

The measurement z in time step t is given by:

$$z_t = Hx_t + v_t \tag{2.2}$$

with

- z_t is m-dimensional vector.
- v_t is measurement error which is assumed to have Guassian distribution $N(0, R)$, with R is a $m \times m$ covariance matrix.
- `H` is an $m \times n$ matrix that associates the state x to the measurement z.

In general, the Kalman Filter algorithm can be divided into two big steps: time update (or "predict") and measurement update (or "correct") [31] (Figure 2.9). The former estimates the state at current time, the latter obtains measurement errors and then updates them to the estimate.

1. **Time update**
   The a priori estimate of the state x_t^- which is the estimate right before the new measurement is computed by the equation:

$$x_t^- = Ax_{t-1} + Bu_t \tag{2.3}$$

   The a priori estimate of the error covariance at time step t is given by:

$$P_t^- = AP_{t-1}A^T + Q \tag{2.4}$$

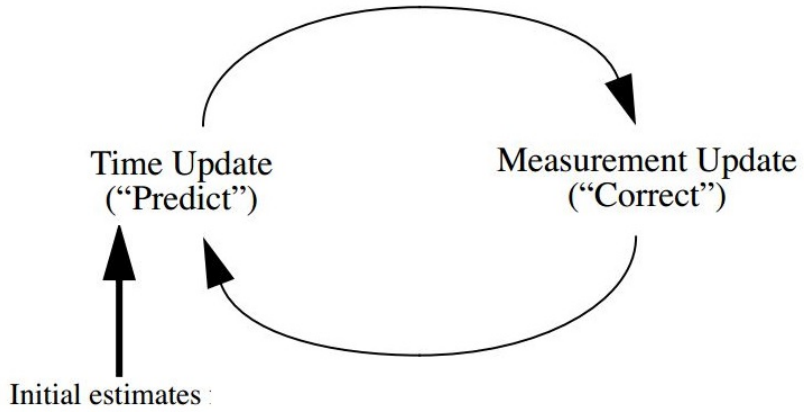


Figure 2.9: The operation of Kalman Filter [31]

2. **Measurement update**
   The following equation calculates the Kalman gain which is then used to find the weight of new information against the already known one:

$$K_t = P_t^- H^T (HP_t^- H^T + R)^{-1} \tag{2.5}$$

   Using the gain and the new measurement z_t to update the estimate of the state:

$$x_t = x_t^- + K_k(z_t - Hx_t^-) \tag{2.6}$$

   Updating the estimate of the error covariance:

$$P_t = (I - K_t H)P_t^- \tag{2.7}$$

   where I is the identity matrix.

The set of equations 2.3, 2.4, 2.5, 2.6 and 2.7 completes the picture of Kalman Filter algorithm. In this report, we are using Q as a constant, but in reality, there are sometimes the case that the process noise covariance Q is changed during the time Kalman filter is working - becoming Q_t. In that case, we need to choose Q reasonably, so that it takes into account the uncertainty of the system.

2.4 Particle Filter

Particle filters, also known as Sequential Monte Carlo methods (SMC), are a class of simulation-based methods which offer a convenient approach to estimate the posterior density distribution of the state-space model.

The filters do not require a fixed functional form of the posterior, such as Gaussian distribution, but only the information of periodic measurement of true state. A finite number of particles are used to approximate posteriors. Those particles are random samples which are drawn from the given density distribution and each is assigned an importance weight to represent its possibility to be sampled from the density distribution. The quality of the approximation depends on the number of used particles, higher number gives a better approximation but more calculation.

By Gordon et al. [13], Particle filters realize the Bayesian theorem which is mathematically stated by the following recursion equation:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} \tag{2.8}$$

where X represents the state of the system and Y is the observation or measurement.

- `P(X)`: the probability density prior to taking measurements.
- `P(Y)`: the evidence or the overall probability of measurements.
- `P(X|Y)`: the posterior probability density of state X with the given measurement Y.
- `P(Y|X)`: the probability of measurement Y in the state X.

The filter implements equation 2.8 by sampling N particles from prior distribution $i = 0, 1, 2, ..., N-1$, each has a state vector X_i.

Then, to compute $P(Y)$, we apply the law of Total Probability:

$$P(Y) = \sum_{i=0}^{N-1} P(Y|X_i)P(X_i) \tag{2.9}$$

The equation 2.9 turns into:

$$P(X_i|Y) = \frac{P(Y|X_i)P(X_i)}{P(Y)} = \frac{P(Y|X_i)P(X_i)}{\sum_{i=0}^{N-1} P(Y|X_i)P(X_i)} \tag{2.10}$$

Each particle is assigned an importance weight w_i to represent its possibility to be sampled from the density function. In other words, w_i describes how closed the particle is to the true state of the system with the given measurement Y, hence we have $w_i = P(Y|X_i)P(X_i)$.

Inserting w_i to equation 2.10:

$$P(X_i|Y) = \frac{w_i}{\sum_{i=0}^{N-1} w_i} \tag{2.11}$$

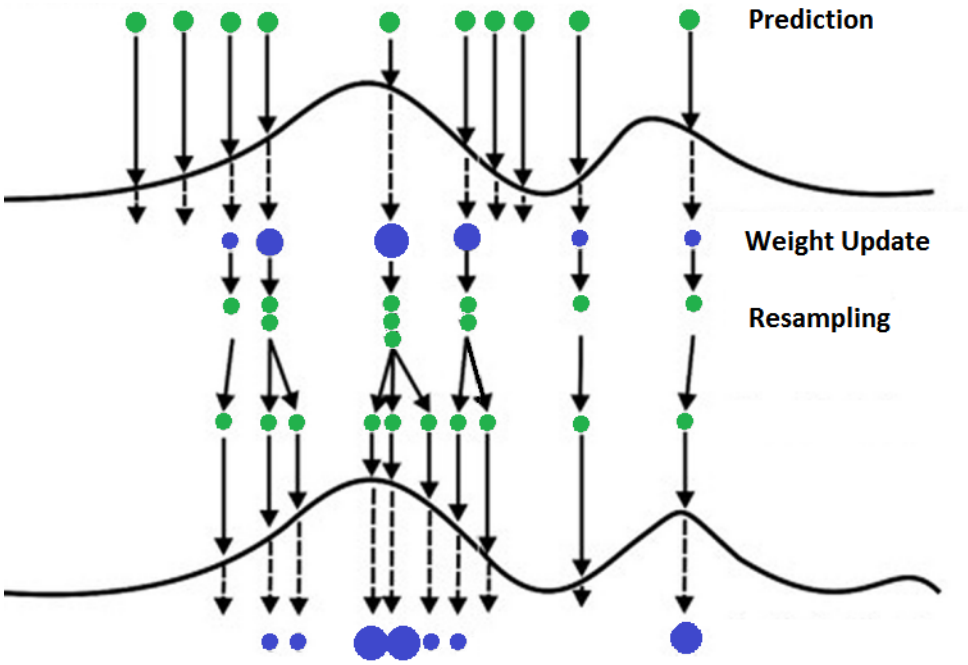


Figure 2.10: Illustration of Particle filter iteration

Equation 2.11 describes simply how a Particle filter is implemented. Basically, it consists of four consecutive steps:

- **Initialization**: in this step, initial particles with equal weights are generated from the given density distribution.
- **Prediction**: Particle filter predicts next positions of the particles based on the expected change of the true state.

- **Importance weight update**: each particle will be evaluated in the new position. Its importance weight will also be calculated and updated according to an error function.
- **Resampling**: this step is the most important part of a Particle filter. It is introduced to avoid the degeneracy problem and to guarantee the convergence of the algorithm. Thus, particles with high importance weight have a tendency to be kept in the next iteration of the filter, while ones with low weights tend to be discarded.

The Particle Filter algorithm is summarized in pseudo code by Thrun et al. [28] as shown in Algorithm 1. Here X_t is the set of M particles

$$X_t := x_t^{[1]}, x_t^{[2]}, \cdots, x_t^{[M]} \tag{2.12}$$

and u_t, z_t are respectively the control and the measurement at time t.

Algorithm 1: Particle Filter Algorithm [28]

$\bar{X}_t = X_t = \emptyset$
for $m \leftarrow 1$ **to** M **do**
  `// Prediction`
  sample $x_t^{[m]} \sim p(x_t|u_t, x_{t-1}^{[m]})$
  `// Importance weight update`
  $w_t^{[m]} = p(z_t|x_t^{[m]})$
  $\bar{X}_t = \bar{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
end
`// Resampling`
for $m \leftarrow 1$ **to** M **do**
  draw i with probability $\propto w_t^{[i]}$
  add $x_t^{[i]}$ to X_t
end
return $\bar{X}_t$

Because of its simplicity in setup and effectiveness in tracking object, Particle filter recently has become a promising tool for lane tracking.

2.5 Cubic Spline Interpolation

Spline

In mathematics, a spline is a numerical function which has intervals between data points are defined by polynomial functions. Thus, spline interpolation is the

process of interpolating each interval of the spline (with the given data points) by a polynomial. The most general form of a *n-degree* polynomial is :

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0 \tag{2.13}$$

To determine all $n + 1$ coefficients of $P_n(x)$, we need at least $n + 1$ different data points.

Cubic spline

Cubic spline is the most popular type of spline because it is widely considered the optimal degree for spline. A cubic spline has control points reside on the curve, therefore they can be used as data points in interpolation. Each interval between control points of a cubic spline is defined by a cubic function. Hence, in the case there are $n + 1$ control points, the function of cubic spline $S(x)$ is:

$$S(x) = \begin{cases} C_1(x), & x_0 \leq x \leq x_1 \\ C_i(x), & x_{i-1} \leq x \leq x_i \\ C_n(x), & x_{n-1} \leq x \leq x_n \end{cases} \tag{2.14}$$

where each C_i is a cubic function.

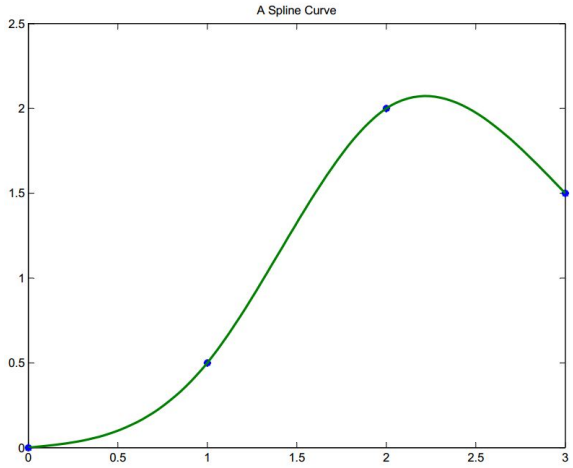


Figure 2.11: Illustration of a cubic spline with 4 control points

The most general form of cubic function which defines the i^{th} interval is:

$$C_i(x) = a_i + b_i x + c_i x^2 + d_i x^3 \tag{2.15}$$

To finish the cubic spline interpolation, it is required to define all the coefficients a_i, b_i, c_i and d_i of each interval. The spline has n intervals, therefore there are totally $4n$ coefficients that need to be determined. In other words, we need at least $4n$ equations.

First, we have the value of spline at each control point:

$$\begin{aligned} C_i(x_{i-1}) &= y_{i-1} \\ \Leftrightarrow a_i + b_i x_{i-1} + c_i x_{i-1}^2 + d_i x_{i-1}^3 &= y_{i-1} \end{aligned} \tag{2.16}$$

and

$$\begin{aligned} C_i(x_i) &= y_i \\ \Leftrightarrow a_i + b_i x_i + c_i x_i^2 + d_i x_i^3 &= y_i \end{aligned} \tag{2.17}$$

Notice that we have $2n$ equations of this type.

Next, conditions to make the spline smooth at the control points are:

$$\begin{aligned} C_i'(x_i) &= C_{i+1}'(x_i) \\ \Leftrightarrow b_i + 2c_i x_i + 3d_i x_i^2 &= b_{i+1} + 2c_{i+1} x_i + 3d_{i+1} x_i^2 \end{aligned} \tag{2.18}$$

and

$$\begin{aligned} C_i''(x_i) &= C_{i+1}''(x_i) \\ \Leftrightarrow 2c_i + 6d_i x_i &= 2c_{i+1} + 6d_{i+1} x_i \end{aligned} \tag{2.19}$$

We have equation 2.18 and 2.19 at all internal points $x_1, x_2, ..., x_{n-1}$. Therefore, there are $2(n-1)$ equations of this type.

Finally, we use the conditions called *natural boundary conditions* at two end points:

$$C_1''(x_0) = C_n''(x_n) = 0 \tag{2.20}$$

By combining the equations 2.16, 2.17, 2.18, 2.19 and 2.20, we have a system of equations with respect to the unknowns are $4n$ coefficients. This system of equations can be solved easily to get the function which exactly defines the cubic spline.

2.6 Hough Transform

The Hough Transform is a popular and relatively fast method for finding simple mathematical forms such as lines and circles in an image. It was originally developed by Hough [16] (1962) to recognize straight lines in a binary image, and later has been developed to detect more general shapes such as curves in [3] and [10]. The basic idea behind the Hough line transform is that each point in the binary image can belong to some sets of possible lines [8]. In general, each line is represented uniquely by a slope a and an intercept b in the slope-intercept form:

$$y = ax + b \tag{2.21}$$

We consider a point (x_0, y_0) in the original image that belongs to line 2.21. Now, by changing a and b, we have a set of many lines passing through the point (x_0, y_0). Thus, moving to the plane (a, b), the point (x_0, y_0) is transformed into a locus of points (here, become a curve) which corresponds to the above set of lines. If we apply this procedure to all nonzero pixel of the original image and accumulate all such curves, the local maxima in the output (i.e., (a, b) plane - commonly called *accumulator plane*) will represent lines in the input (i.e., (x, y) plane).

The slope-intercept form however is not able to represent vertical lines. Hence, the preferred representation is polar form (ρ, θ) with θ is the angle of the line and ρ is the distance from the line to the coordinate origin. The equation of polar form is:

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{2.22}$$

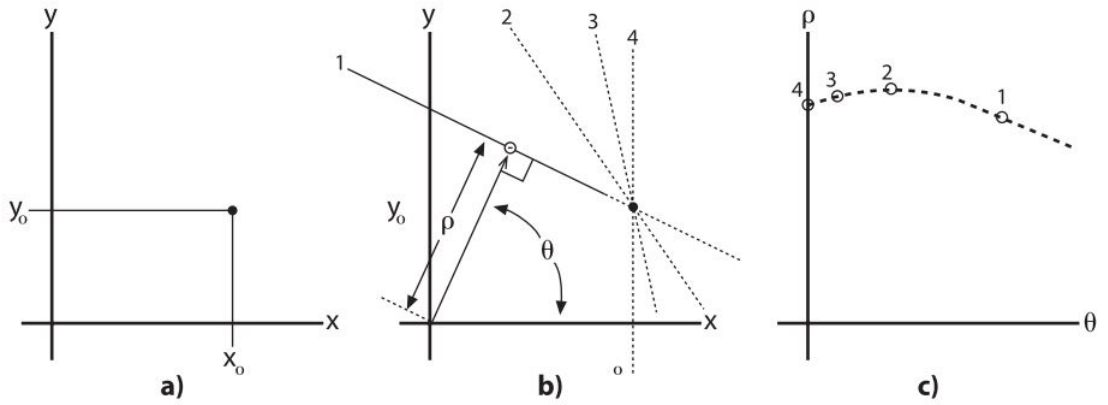


Figure 2.12: A point in the original image (a) can be passed through by many lines (b), each is parameterized by a different (ρ, θ). In (c), these lines are represented by the points in the (ρ, θ) plane. [8]

The line detecting algorithm with the Hough Transform can be divided into several steps:

1. Edge detecting.
2. For each edge point, we can define a set of lines passes through it and map it to the Hough space ((ρ, θ) plane) as a curve. An accumulator is used to store those curves.
3. Several different methods such as thresholding can be applied to extract the local maxima from the accumulator. These maxima represent straight lines of the original image.

3 State of the Art in Lane Detection

This chapter describes the common system flow of almost all lane detection systems. We also discuss several different theoretical approaches which are current state of the art in lane detection research.

3.1 Initial Concerns in Lane Detection

Before jumping into the details of including steps in lane detection algorithms, there are several initial concerns that we firstly need to consider to discover an appropriate approach. These concerns include system objectives, working environment and sensing modalities.

3.1.1 System Objectives

The direct information provided by a lane-position detection system is the relative positions of lane boundaries with respect to the position of the vehicle. From this information, we may extract several useful inputs, such as lane crossing point, lane keeping performance, etc., to many kinds of automotive safety systems.

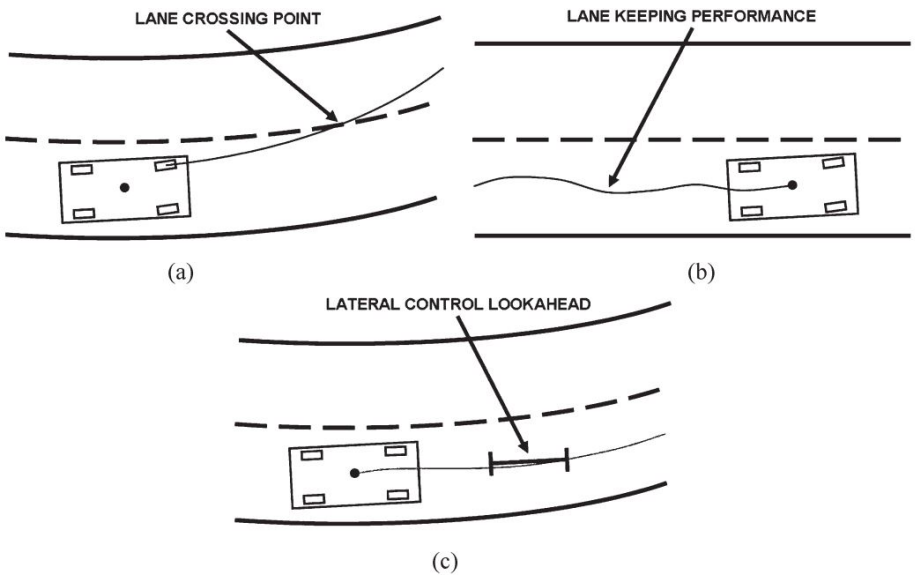


Figure 3.1: Illustrations of main objectives in lane-position detection [22]. a) Lane Departure Warning; b) Driver Attention Monitoring; c) Automated Vehicle Control.

In [22], McCall and Trivedi describe three main objectives of lane detection algorithms:

- **Lane Departure Warning System**: Based on the information about lateral deviation of the vehicle to the lane's center, speed, etc., we have to predict the trajectory of the vehicle: whether it is leaving the lane or not and where the crossing point is.
- **Driver Attention System**: In monitoring task, there are many factors that need to be considered to manage the attentiveness of the driver. One important information can be provided by lane detection algorithms is the smoothness in lane keeping.
- **Automated Vehicle Control System**: This is the most complicated one in all three objectives and recently has received a considerable attention from researchers around the world. It requires from lane detection algorithms the accurate lateral deviation at the moment and predictions in distance ahead.

For a specific objective, different algorithms and sensors will have different performances. That is why we have to consider carefully our requirements to the lane detection algorithm, what role we want it to play in a bigger system, and then we can choose an appropriate approach for our needs.

3.1.2 Working Environment

Besides the system objectives, there is another important concern in lane detection we need to pay attention to, which is the working environment of the algorithm, or in the other words, which scenarios will our system have to deal with? Characteristics of roads and road markings can vary greatly depending on such diverse scenarios, for example in urban streets, suburb streets, highway, etc. In figure 1.1, we provide illustrations of several different scenarios that lane detection algorithms might have to deal with in reality.

Even after taking into consideration that our system only works in a particular scenario, there are still many things that need our attention. The performance of the system is changed even on the same road if it works in different light conditions and traffic intensities. Color of the roads can alter significantly within a short distance. Additionally, road surface may consist of slopes with different angles or use different types of barriers. Lane markings can also vary greatly on the same road. They can be in many sizes and different colors such as white or yellow, or combination of solid lines, long and short dash lines, circular reflectors, etc. Figure 3.2 illustrates the variation of road and lane marking on a same highway in Korea. Figure 3.2(a) shows the change in color of the road surface and left lane marking in yellow color. Figure 3.2(b) shows where the road branches with a turn-right sign and Korean

letters on the surface. Figure 3.2(c) depicts the place where the right road marking is faded. Figure 3.2(d) shows a road marking which is a combination of different types.

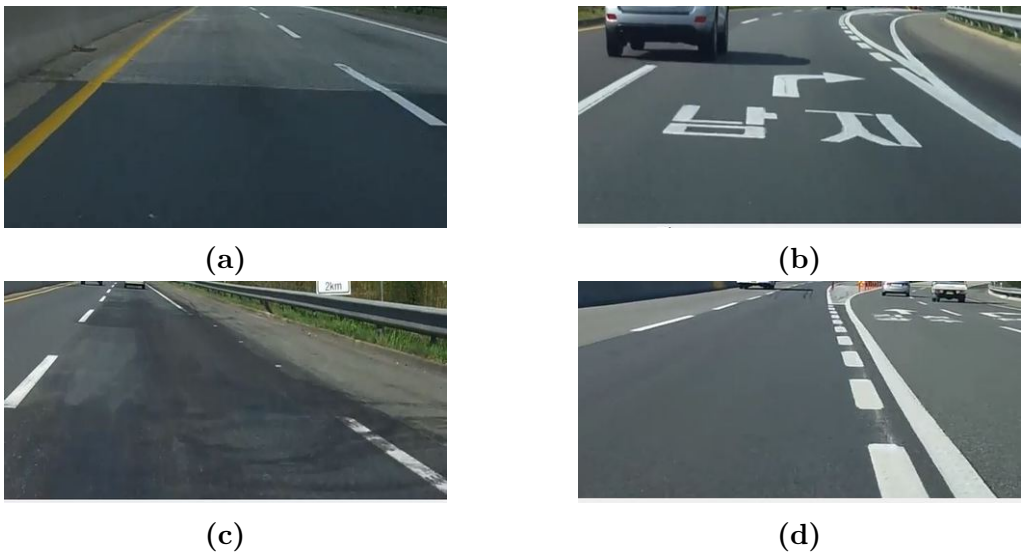


Figure 3.2: Illustration of the variation of lane markings in a road

3.1.3 Sensing Modalities

Many different types of sensors have been used to provide inputs for lane detection algorithm, some of them are cameras, Global Positioning System (GPS), RAdio Detection And Ranging (RADAR) sensor, LIght Detection And Ranging (LIDAR) sensor, etc. Each type of sensors provides different kind of information such as vision, distance and position. That is why a sensor can work well in a certain condition but fail in others. The following table shows the advantages and disadvantages of each type of popular sensors:

Sensors	Advantages	Disadvantages
Vision sensors	This type of sensors can work well independently in a large variety of situations without requiring the aid from external infrastructures or map data.	Vision sensors cannot perform well in situations where lane markings are totally covered or faded or the system is working in critical light conditions.
Internal vehicle-state sensors	Playing the only role of providing supporting information for other types of sensors.	This type cannot be utilized independently but has to collaborate with other sensors in the system.

Line sensors	This type of sensor can provide accurate value of current lateral deviation of the vehicle.	It cannot provide information in distance ahead which is used to predict trajectory of vehicle.
LIDAR and RADAR	These sensors are utilized to detect objects and calculate distance. They are very useful in defining the road boundaries, especially in rural areas where there are no marked roads.	These sensors do not perform well in lane detecting, so most of them require the assistance of vision sensor.
GPS and digital maps	Enhanced type of GPS (Differential GPS) can provide location of vehicle with the accuracy of about $10cm$, therefore it is very useful in monitoring the vehicle's movement.	This sensing modality requires proper implementation and support from a modern external infrastructure to achieve good enough accuracy. Additionally, digital map data have to always be updated.

Table 3.1: Advantages and disadvantages of different types of sensors

Because of its wide utilization, vision has become a notable research area in lane detection. This thesis will focus on the lane detection algorithms that use input data from one monocular vision camera.

3.2 General Model of Lane Detection Algorithms

In 2006, after taking an extensive survey in many lane detection algorithms that have been developed, McCall and Trivedi [22] proposed a common diagram to generalize the system flows of those algorithms (Figure 3.3).

This flowchart can still be applied for almost all lane detection algorithms until now. It includes several steps:

1. First, a system of input sensors is used to collect the information about surrounding environment. As described in Section 3.1.3, there are various types of sensors that can be used and each provides a different kind of data, for example camera - the most popular type or GPS in [19] or LIDAR in [9]. However, in this thesis, we focus only on the methods using input data from one monocular vision camera.

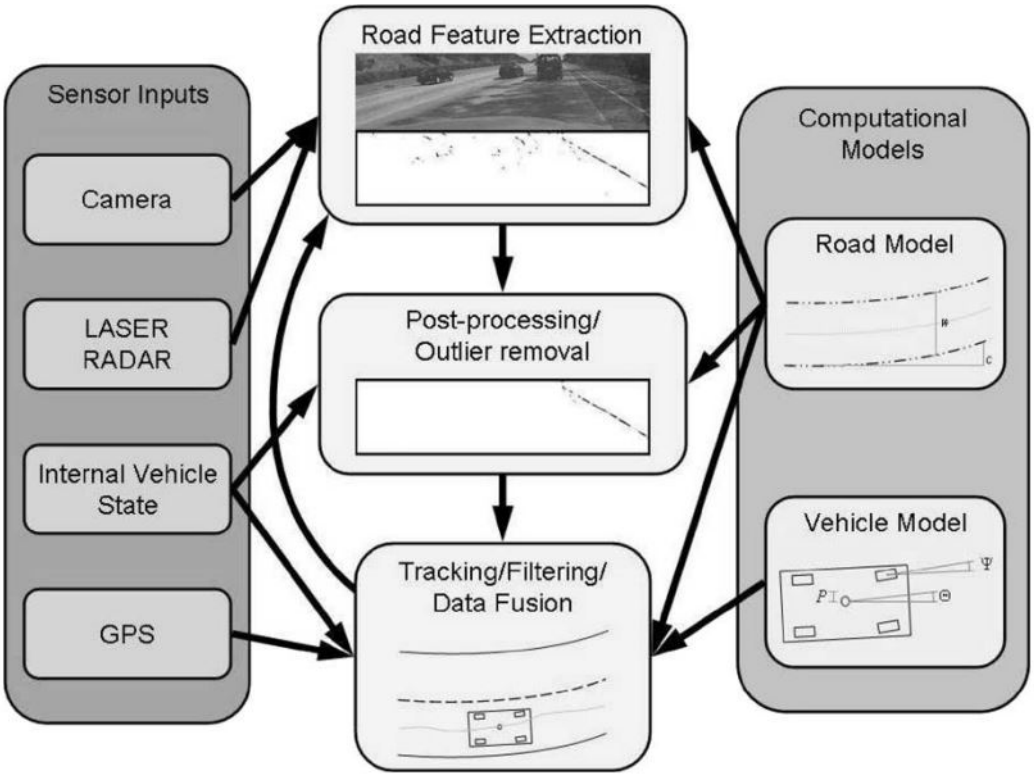


Figure 3.3: Generalized flowchart of lane detection systems [22]

2. Next, a model of roads or lane markings is proposed. It could be a simple straight line as in the research of Apostoloff and Zelinsky [2] or a more complex model like spline in [4] and [30].

3. The input data is then processed with the aid of the proposed road model to extract the information about road features such as edges, textures.

4. Depends on the scenarios we are dealing with, a linear approximation or a real steering motion model can be utilized to improve the estimation of tracking stage.

5. Finally, a tracking stage can be applied to help in predicting and smoothing. Two most popular techniques used in this stage are Kalman Filter as in [7] and [29] and Particle Filter in [4].

In general, most of algorithms until now follow this common flow. But depends on the objectives and working scenarios, there may be variations between them, for example several systems like [1] and [5] works only on single frame instead of using lane tracking stage.

3.3 Comparison of the State-of-the-Art Approaches

Comparison of the current state-of-the-art researches is based on the general flowchart in Figure 3.3. There are four main points that all researches pay attention to, which are road modeling, lane feature extraction, post-processing/lane detection and lane tracking. This section discusses the advantages and disadvantages of each approach with respect to those points.

3.3.1 Road Modeling

Most approaches propose at least one road model, either simple or complex, to represent lane marking. Road modeling can help significantly in boosting the performance of lane detection system by eliminating the outliers and ignoring the evidences created by noises. Several road models mainly used in all studies are discussed in this section.

Straight Line

A straight line is the most simple model that can be used to represent lane marking. Although its representing ability is limited, it has the advantage of low computation time. The approaches that use straight line model mostly work in roads with smaller curvature, due to the fact that they are based on a simple assumption, which is lane marking in Region Of Interest (ROI) can be approximately described by a straight line.

Several researches tries to find straight lines in the perspective binary image such as Kuk et al. [18] and Voisin et al. [29] (Figure 3.4).

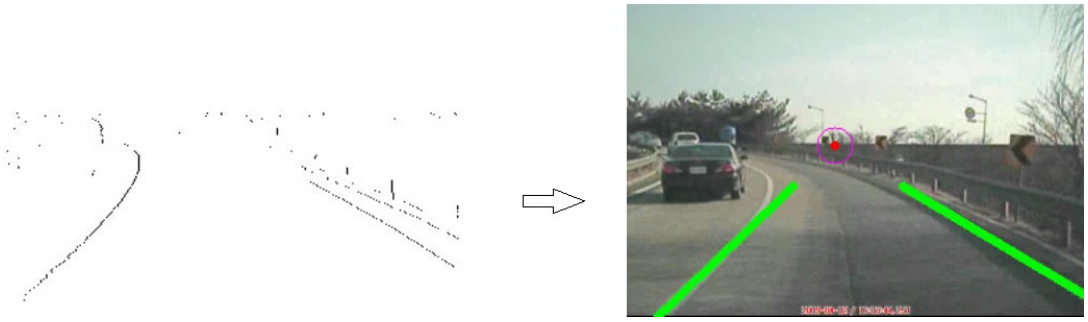
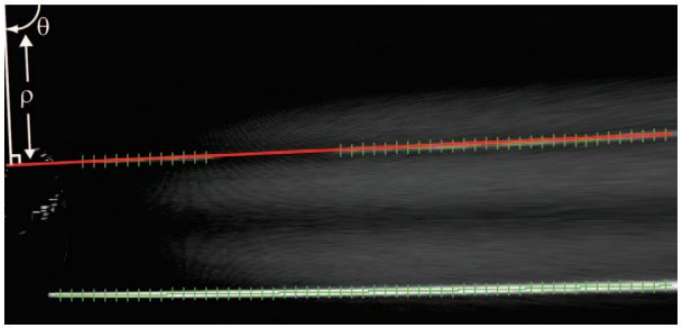


Figure 3.4: A simulation result from [18]

Some other approaches such as Borkar et al. [7] and Bertozzi and Broggi [5] convert the original image to IPM image (bird's-eye view) to reduce the curvature of lane marking. The line detection is then carried out in IPM image. Here, the assumption of a flat road is shared among these researches.

(a) Camera perspective view **(b)** IPM image (bird's-eye view)

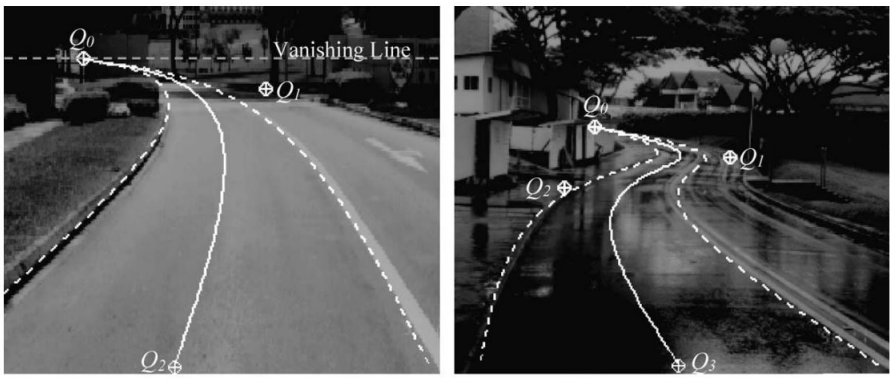


(c) Fitting lines in IPM image

Figure 3.5: Using straight lines to represent lane markings in IPM image [7]

Curve

Because of the limitation of straight lines in describing lane marking, especially in the case where roads have high curvature, the recent researches focus on the method of using curve as lane-marking representation. There are many kinds of curves that may be utilized as lane-marking model such as Bezier spline, cubic spline, clothoid, etc. In general, these curves are defined by choosing 3 or 4 control points in a specific way, for instance, the approaches described in researches [1], [4], [17] and [30]. The number of control points used can be explained by the same reason as the overfitting problem in regression analysis. Using too many control points increases not the representing ability, but the computation time.



(a) Using 3 control points (b) Using 4 control points

Figure 3.6: B-Snake based lane model [30]

In a famous research published in 2004, Wang et al. [30] proposed using B-Snake model for lane representation: *"B-Snake can describe much wider range of lane shapes while retains compact representation, since B-Spline has local controllability and can form arbitrary shape."* [30] (see Figure 3.6). B-Snake model is considered to be robust against shadow, noises, faded or lost lane marking, etc.

In 2008, Kim [17] published a highly appreciated research which uses cubic spline with the control points located on the curve to represent lane marking. In his research, Kim compares his model to the B-Snake model of Wang et al. [30] and points out: *"In a B-spline representation, control points reside outside of the curve, and its fitting procedure requires a significant number of iterations. On the other hand, the uniform cubic-spline fitting is much faster..."* [17]. Under testing, Kim's method runs faster and shows a better result than Wang et al.'s in most cases, but does not performs as well in the cases of missing or false lane markings.

Discussion

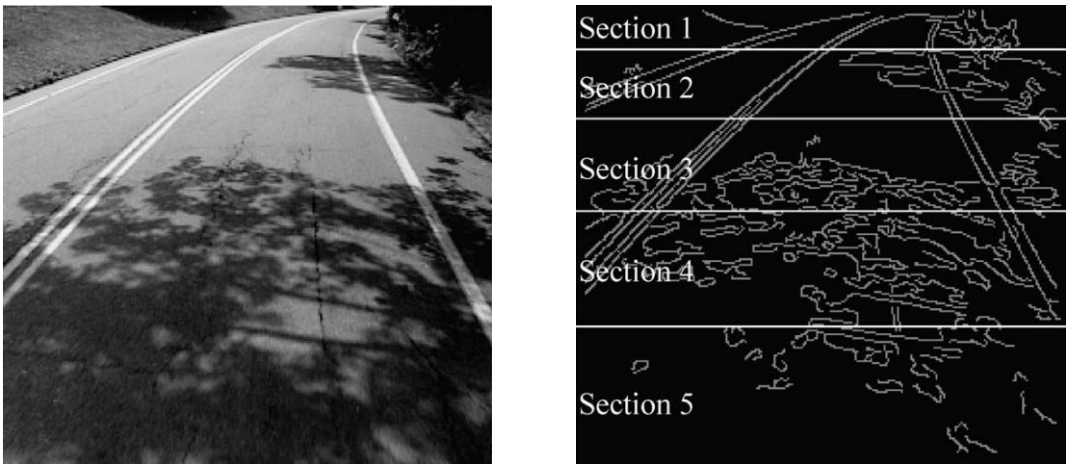
By the advantages and disadvantages of each lane model that are described in the above researches, it is hard to say which model is the best for all cases. The choice of lane model depends on the situation that the lane-position detection system has to deal with. For example, highway roads normally have fairly simple structure, therefore it is more reasonable to use simple models. Besides, if the vehicle is moving in low speed, the safety system might require only about $10m$ of the road ahead, therefore a linear lane model can satisfy the requirements in this case. But considering a more complex situation, for the system like Lane Departure Warning System or Automated Vehicle Control System, it is necessary to predict precisely the trajectory of vehicle in the next at least few seconds. In the situation of highway road, the vehicle is moving in high speed, the concern should be at least $30 - 40m$ ahead. In this case, a more complex model like parabolic curve or spline would be essential (see Figure 3.6).

3.3.2 Lane Feature Extraction

This is the first stage, which plays an extremely important role, in three main stages of lane-position detection system. While the proposed lane model is only considered to be a supporting step for lane detection process, Lane Feature Extraction is indispensable and can totally change the final result of the whole system. A bad result of lane feature extraction will definitely lead to a bad final result, regardless how good the other stages are. On the contrary, if the extraction stage can perform perfectly, we do not even need a lane model to make the system work well. Similar to road modeling, the choice of lane feature extraction method can also vary depending on the type of system and working environment. This section discusses several different methods used for lane feature extraction.

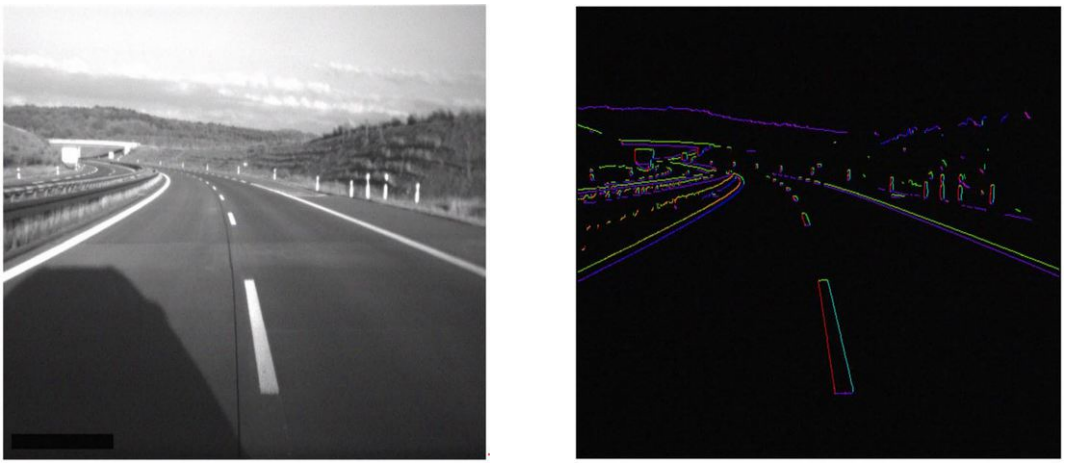
Edge-based techniques

Edge-based lane feature extraction is the most popular approach in all the researches. It is based on a fact that, there is always a contrast in color of lane marking and road surface. Therefore, edge-based techniques are effective in most of cases, especially with solid and dashed lane marking. Here, there are various edge detection techniques in image processing can be utilized for lane feature extraction such as Canny, Sobel, Prewitt, etc. Several auxiliary steps may be additionally applied to increase the effectiveness of edge detection, for example, blurring or thresholding.



(a) Grayscaling image **(b)** Edge-detected image

Figure 3.7: Example of using Canny edge detector by Wang et al. [30]



(a) Grayscale source image **(b)** Resulting edge point image

Figure 3.8: Using Canny edge detector with gradient information in the output by Lindner et al. [20]

Although the edge detection techniques work effectively with fairly good speed in most cases, they still show limitation in more complex situations, for example, Figure 3.7 shows too many false evidences that are left in the image of road with critical shadow condition.

Horizontal intensity-bumped filters

These filters can be considered as the improved version of edge detection for lane feature extraction. They are based on the idea that lane marking in general has vertical direction in ROI, especially in IPM image. Instead of detecting edges in all directions of the image, they compare the intensity value of each pixel to its right and left neighbors. Therefore, the filters produce high responses to the pixel which has higher intensity value than those neighbors. Using the filters is much faster and can help in overcoming the limitation of ordinary edge detection techniques. Several researches use this approach can be listed out [1], [5], [23].

Figure 3.9 shows the method of Aly [1], which uses a two dimensional Gaussian kernel to filter the IPM image. Figure 3.10 illustrates the result of Nieto et al. [23] by applied their customized filter.

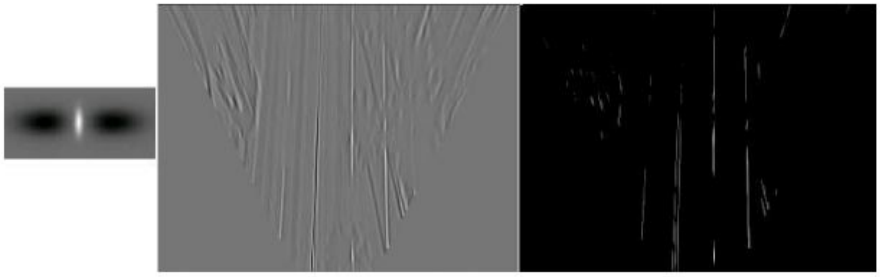


Figure 3.9: Image filtering and thresholding by Aly [1]. Left: the kernel used for filtering. Middle: the image after filtering. Right: the image after thresholding

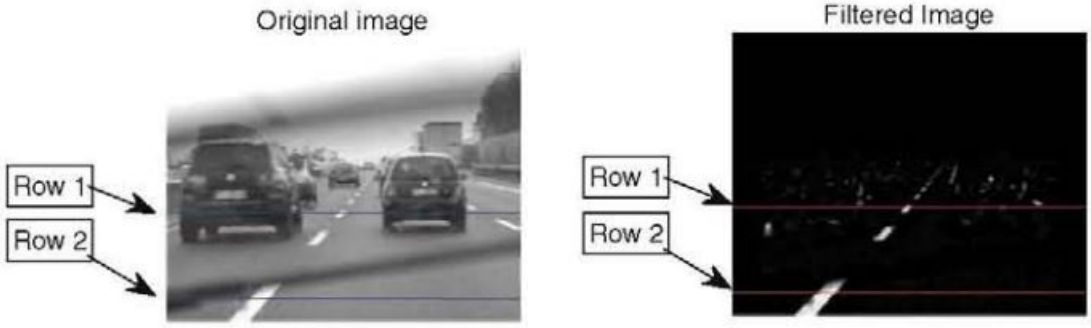


Figure 3.10: Result when applying filter in perspective image of Nieto et al. [23]

Machine learning

Although performing well in many cases, the horizontal intensity-bumped filters still easily confuse false evidences caused by other vehicles or unwanted marks on road surface with the evidences of weak lane markings. To overcome this limitation, some researches propose applying machine learning and the most notable research recently uses this approach is Kim [17]. First, Kim collects different image patches of lane marking and non-markings for learning (see Figure 3.11).

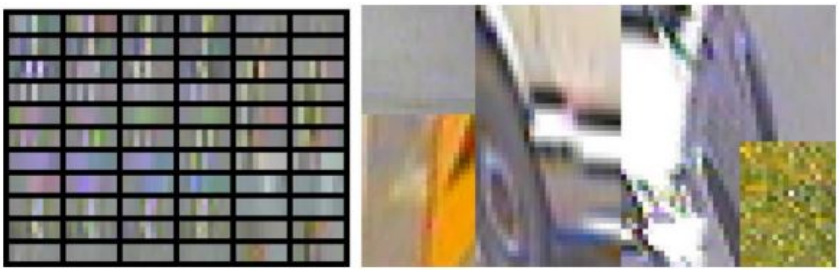


Figure 3.11: Example of image patches of lane markings and non-markings [17]

Then he tests several different machine learning classifiers in IPM image which include: Intensity-Bump Detection, Artificial Neural Networks (ANNs), Naive Bayesian Classifiers (NBC), Support Vector Machine (SVM). The graph in Figure 3.12 and the table in Figure 3.13 show the testing result of Kim for all the classifiers. Based on this result, SVM has the best accuracy but it is not fast enough for real-time classification, therefore finally ANNs with the accuracy in second place and the classification time around $100ms$ is chosen to use.

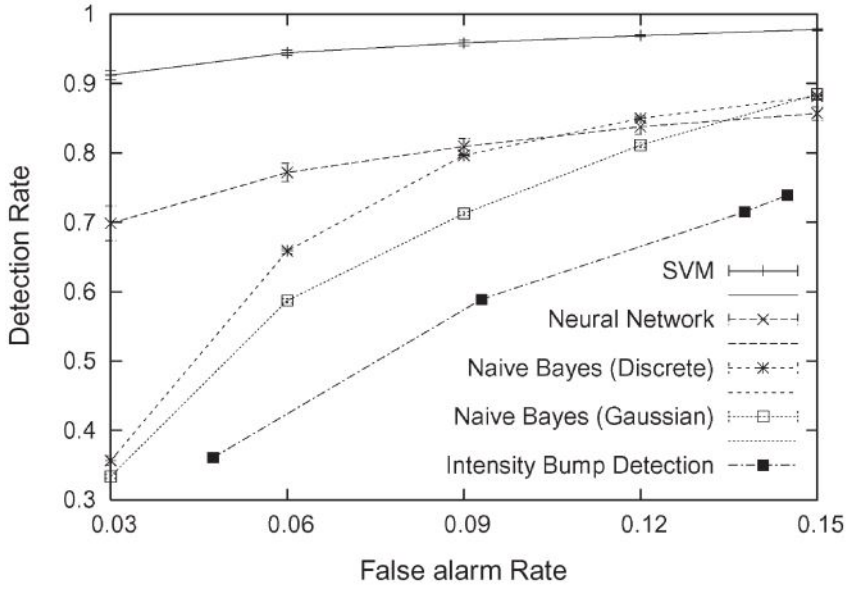


Figure 3.12: Classification performance of the classifiers [17]

Classifier	Classification time
Intensity Bump Detector	~ 10 ms
ANN (7 hidden nodes)	~ 100 ms
Gaussian NBC	~ 25 ms
Discrete NBC	~ 60 ms
SVM	~ 2.2 sec
Cascade	~ 25 ms in worst cases

Figure 3.13: Computation time of the classifiers [17]

The result of Kim is highly appreciated because of its effectiveness and its ability to handle many complex situations. However, the method requires the training set to be chosen carefully in different conditions such as good/bad weather, daytime/nighttime light, etc. to produce a good result.

Stereo vision

Recently stereo algorithms have been used by several researches to further improve lane feature extraction performance. For example, in the case of using left and right camera, by comparing 2 images captured by the cameras, lane markings can be extracted quite easily and the distances in the images can be calculated more precisely. The notable researches that utilize this approach are Danescu and Nedevschi [9], Leonard et al. [19] and Hattori [15]. Although in this thesis, we only focus on algorithms using monocular camera, using stereo camera still is a promising direction for further research in the future.

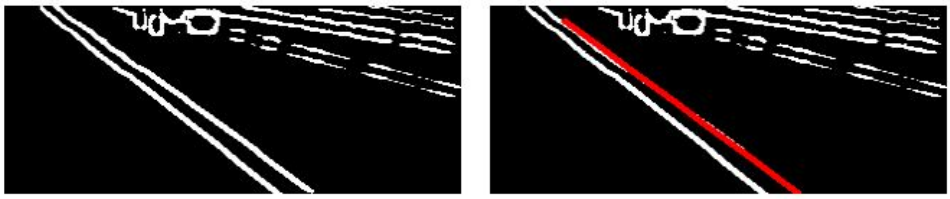
3.3.3 Post-processing

This stage is also called Lane Detection stage. Based on the knowledge about lane markings that we receive after the lane feature extraction stage plus the proposed road model, post-processing stage is necessary to estimate the lane-marking positions. The most common techniques that are used in this stage are Hough Transform and RANdom SAmple Consensus (RANSAC).

Hough Transform

Hough Transform is an extremely popular technique used for lane detecting because of its high speed and effectiveness in line detecting. The researches of Aly [1], Borkar et al. [6] (see Figure 3.14), Voisin et al. [29] and Wang et al. [30], all apply Hough Transform in their approaches. The disadvantage of Hough Transform is that this technique is only useful for straight line detecting in the image, therefore it should be used in combination with other techniques to improve the result. For example, Wang et al. [30] use an algorithm called CHEVP (Canny/Hough Estimation

of Vanishing Points), which is the combination of Canny edge detector and Hough Transform for effectively vanishing points detecting.



(a) Binary image (b) Hough transform for line finding

Figure 3.14: Applying Hough Transform in [6]

RANSAC

RANSAC is a highly effective method for model robust fitting and data outlier removal. In some researches, RANSAC is used in combination with Hough Transform for line or spline fitting, such as the approach of Aly [1] (see Figure 3.15).

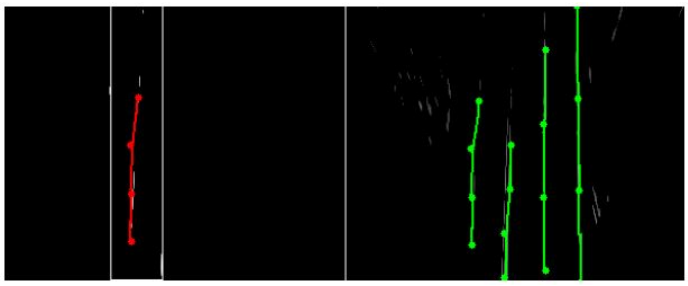


Figure 3.15: RANSAC spline fitting [1]

Kim [17], in his famous research, uses RANSAC to find the hypotheses of lane markings among many noises (Figure 3.16).

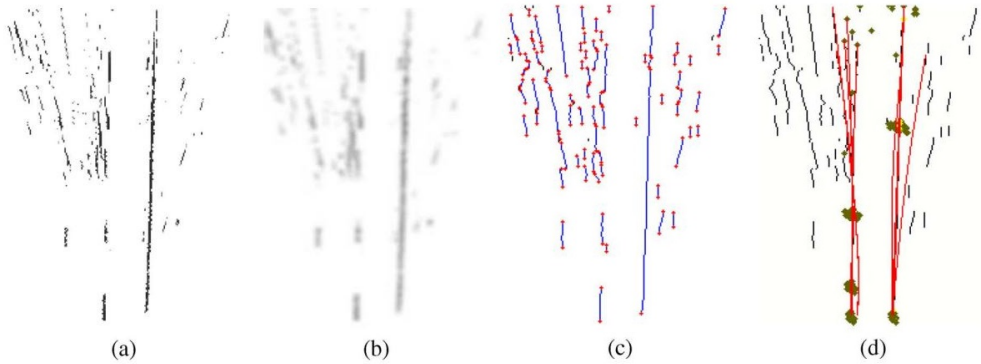


Figure 3.16: Example of using RANSAC by Kim [17]. a) Detected lane features; b) Applying Gaussian smoothing; c) Line-segment grouping; d) Selected hypotheses from RANSAC algorithm.

Random hypotheses

This is a fairly simple and effective method to find the best lane marking hypotheses in the processed image. The idea is to generate many hypotheses of lane marking over the image and assign a weight to each hypothesis to describe its possibility to represent the lane marking. The weight can be calculated through a function based on position of the hypothesis in the image. The number of hypotheses used may be reduced significantly by applying tracking algorithm. The most notable advantage of this method is that the hypotheses later can be used as input for the tracking stage. In the research [17], beside RANSAC, Kim uses Particle Filter to generate another set of hypotheses. Berriel et al. [4] also chooses Particle Filter for lane detecting (see Figure 3.17).

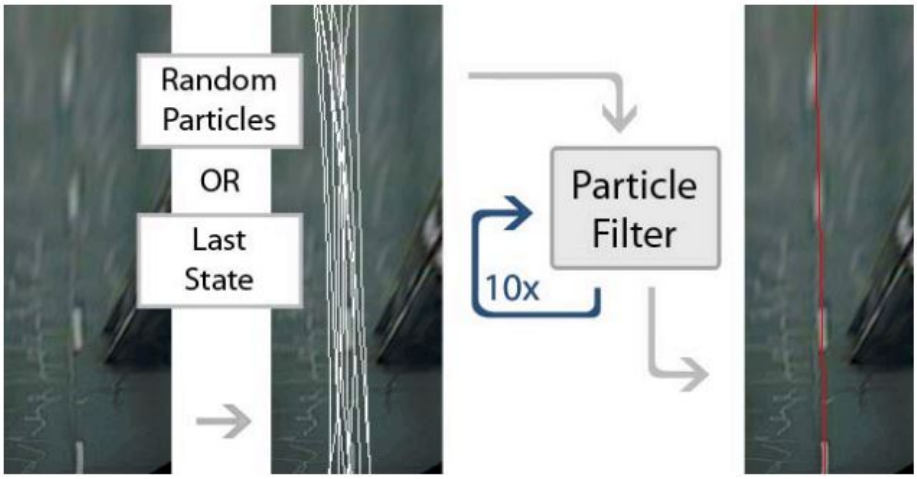


Figure 3.17: Running Particle Filter several times to detect lane marking [4]

Multi level grouping and Classification

In [20], Lindner et al. propose a special approach for lane detection which is based on the resulting edge point images from Canny edge detector (Figure 3.8). This method not only detects evidences of lane markings, but also can classify lane markings into different classes which are the official directives for lane markings in Germany. Each types of roads, like highways or urban streets, has different kind of lane markings. Therefore, this information is, in fact, extremely useful for ADAS system in assessing situations and traffic rules. The basic idea of the method is dealing with different levels of lane marking abstraction, from simple to complex. For example, lines are detected from the edge points in the resulting edge point image, then a fitting value is calculated to group lines into curves, and so on, to finally get closed objects (Figure 3.18).

(a) Result image of grouped objects **(b)** Grouped objects in source image

Figure 3.18: Result of multi level grouping in [20]

3.3.4 Lane Tracking

The lane tracking stage is not always used in all researches, for example the systems of Aly [1] and Bertozzi and Broggi [5] work on single frame instead of using tracking stage. However, in most researches recently, by applying tracking algorithms, the speed and accuracy of the systems have been improved significantly. Normally, tracking algorithms are used in the combination with lane feature extraction to form a closed-loop feedback system. The two most popular tracking algorithms used in lane-position detection system are Kalman Filter and Particle Filter, whose theories are presented in Section 2.3 and 2.4. In this section, we only discuss their applications in some studies.

Kalman Filter

Working under a strong set of assumptions, Kalman Filter is able to handle relatively well most of the common cases. Besides, it has the massive advantage of computation time and speed. This is the reason why Kalman Filter is chosen to use in many researches such as McCall and Trivedi [22], Borkar et al. [7], Danescu and Nedevschi [9], Voisin et al. [29], etc.

However, in reality, a lane-position detection system has to deal with many complex situations, for example road has high curvature, noises left after lane feature extraction, false or missing evidences of lane markings, etc. In these cases, the result of Kalman Filter will become inaccurate after only few iterations.

Particle Filter

The result of lane position sometime needs to be extremely precise so that it can be provided to safety system of vehicle. Therefore, to overcome the limitation of Kalman Filter, some researches choose to use Particle Filter for lane tracking. The

effectiveness and speed of Particle Filter depend on the number of particle used. In general, Particle Filter requires more calculations compared to Kalman Filter, but it is much more effective and less prone to noises.

Several notable researches that apply Particle Filter are [2], [4], [17], [26]. Among them, Berriel et al. [4] and Kim [17] use Particle Filter for both lane detecting and lane tracking, while the others utilize the results of lane detecting stage as the input for Particle Filter.

3.3.5 Common Assumptions

The assumptions are an important part of a real-life problem such as lane-position detection. They are necessary because they are helpful not only in improving the overall performance of lane-position estimation, but also in restricting the scope of the problem that the algorithms have to deal with, so that we know, out of that scope, the systems may fail. The common assumptions are summarized by McCall and Trivedi [22] as below:

- Texture of the road or lane is considered to be nearly constant.
- Width of the road or lane is locally constant.
- Appearance and placement of lane markings follow strict rules.
- The road surface is considered to be almost flat or has elevation change follows a strict model.
- There is a limit for the road/lane curvature.

All of the available lane-position detection algorithms until now use at least one or more of the above assumptions, therefore they may focus on solving only a specific problem. However, to take the full advantage of the assumptions, their role in lane-position detection problem needs to be well understood and used reasonably.

3.4 Conclusion

This chapter presents current state of the art in lane detection research. The general flowchart, which is followed by most of the existing researches, includes three main stages: lane feature extraction, post-processing/lane detection and lane tracking. Besides, a road model and a vehicle motion model are proposed to provide more information for the calculation and estimation.

There are a huge number of algorithms that can be applied to deal with the lane detection problem. However, firstly there are three main factors that need to be considered in each research before deciding which algorithm should be used. They include system objectives, working environment and sensing modalities. Under a careful consideration of these factors, each research will have a different approach to the problem.

The chapter examines each approach regarding four main points: road modeling, lane feature extraction, post-processing and lane tracking. For each point, several most popular used methods are presented and compare with each other to identify their advantages and disadvantages. Finally, we introduce the most common assumptions used by the researches in lane-position detection.

Based on the in-dept research into state-of-the-art lane detection algorithms in this chapter, three different algorithms have been constructed and implemented in our available hardware. To guarantee the working speed of the system, we try only the straight line and cubic spline model to represent lane marking. Then a fast and effective filter is applied to extract lane feature. Besides, the techniques of Hough Transform and random hypotheses are chosen for post-processing. Finally, for lane tracking, both Kalman Filter and Particle Filter will be tested. The details of each algorithm are presented in the next chapter.

4 Concepts

After an in-depth research on the current state-of-the-art work in lane detection, we made a comparison among them and, based on the requirement of the thesis on a system working real-time in a low power embedded hardware, implemented three different lane-position detection algorithms in our real hardware - Raspberry Pi 3 - to evaluate their performances. This chapter describes in detail the concepts and workings of each algorithm.

4.1 Overview

The general flow of all three algorithms in the thesis is represented in Figure 4.1.

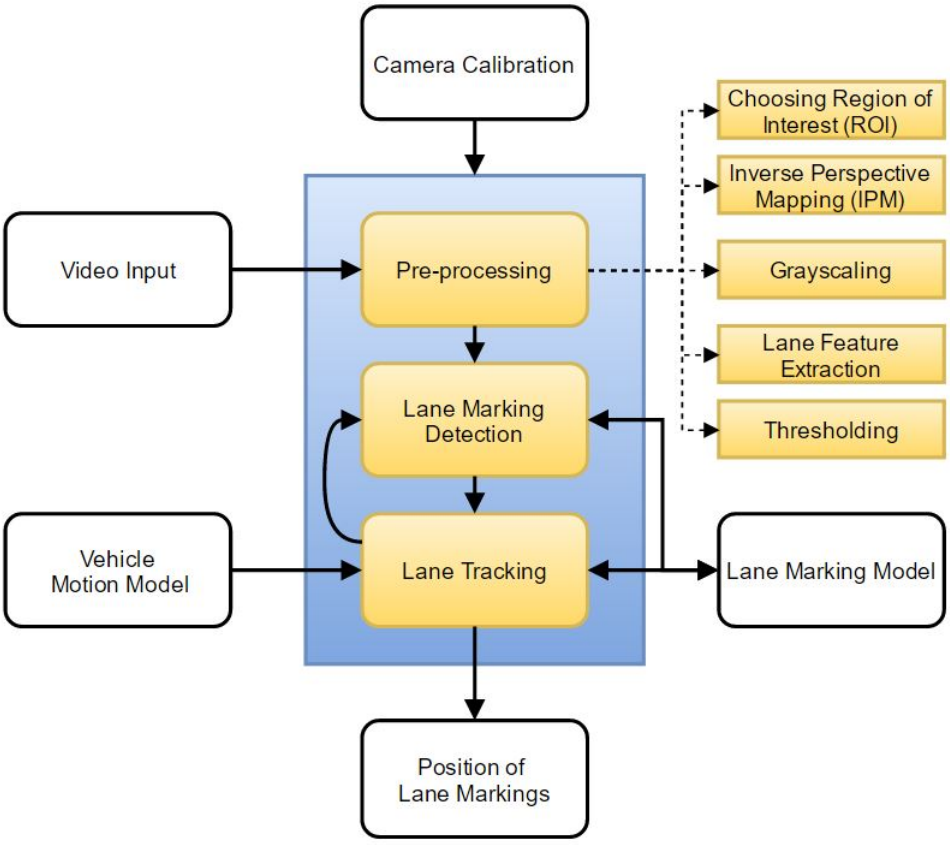


Figure 4.1: General workflow of approaches in the thesis

Before going to the main stages of the system workflow, we need to mention several important supporting models. First, camera calibration is not a part of the working of the system but for all computer vision systems, it actually has an extremely large influence on the result. The parameters include position, view angle and the internal parameters of the camera. The internal parameters of a camera are the position of image center on the image, focal length, scaling factors and lens distortion. Defining these parameters correctly can help significantly in increasing the accuracy of the system. Most researches do not mention this step because it is the same and compulsory in all situations. Only in the research of Wu et al. [32], the team has proposed a method which is applied to dynamically calibrate camera parameters to adapt to different situations. We also do not discuss camera calibration in the thesis. Second, because of the limitation of hardware power, we only use straight line or cubic spline for lane marking model. The detail of this choice will be discussed in following sections. Third, the vehicle motion model will be represented by a random variable with normal distribution. In the common scenarios, this assumption does not significantly reduce the accuracy of the final result.

The workflow includes three main stages:

1. **Pre-processing**: The input data of the system, which can be a video or a stream of image frames from a dashboard camera, will go directly into this stage. Each frame of image is then processed to extract the evidence of lane markings and provide to Lane Marking Detection stage. The Pre-processing stage may consist of five phases: choosing the ROI, IPM, grayscaling, lane feature extraction, thresholding. In our three proposed algorithms, Algorithm 1 and 3 do not use IPM in their Pre-processing stage.

2. **Lane Marking Detection**: this stage finds the positions of the lane markings in the processed image provided by Pre-processing stage with the aid of the information about lane-marking model. Normally, the Lane Marking Detection stage performs whenever the estimation of the lane markings is not available, for example in the very first frame of video stream. Besides, this stage consumes a lot of resource of the system, therefore, it only runs when the tracking stage is not able to track lane markings.

3. **Lane Tracking**: this stage uses a tracking algorithm such as Particle Filter to track the positions of lane markings in processed image. The main difference of Lane Tracking and Lane Detecting stage is that, Lane Tracking stage uses estimation from the previous frame to predict the position of lane markings in the current frame. Hence, this stage performs whenever the information from the previous frame is available. In general, because of its low resource consumption, Lane Tracking stage is important in helping increase speed and performance of the system and in any cases, we try to use Lane Tracking as much as possible.

During the implementation process, in each stage, we have tried many different algorithms for the purpose of testing and evaluating. The thesis only describes the best result we have. In the next three sections, three algorithms that have been implemented will be explained in detail.

4.2 Line-based Lane Detection Algorithm

In the first algorithm of this thesis, to guarantee a high working speed for the system, we choose the approach that uses simple straight lines to represent lane-marking model. This straight-line choice is based on the assumption that, in common scenarios, the lane markings appear on ROI do not have high curvature and can be adequately approximated by straight lines. After that, Lane Detection stage is realized by a technique called line normal random sampling. Finally, a Particle Filter is exploited to track the positions of lane markings in tracking stage.

4.2.1 Pre-processing

The pre-processing stage extracts the evidences of lane markings from the raw input image to provide to the next stage. In this first algorithm, pre-processing stage includes four smaller consecutive steps: choosing the ROI, grayscaling, lane feature extraction and thresholding. The flow of the stage is illustrated by Figure 4.2.

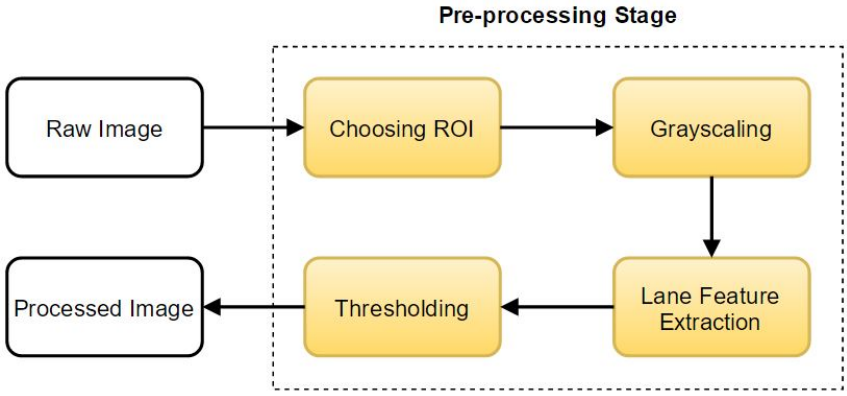


Figure 4.2: Flow of Pre-processing stage in the first algorithm

Choosing Region of Interest (ROI)

The choosing ROI step crops the raw image to a specific area, which likely contains the evidence of lane markings. This very first step is necessary because of the fact that, in general, the road region is quite small compared to the whole image. This step simply removes unexpected noises from other things like sky, pavements or trees and therefore, it is important to increase the speed of the system by reducing the amount of data that needs to be processed.

Figure 4.3: Example of choosing ROI step

Grayscaling

This step converts the ROI image in RGB color format to grayscale format. Normally, the input image is provided in RGB color format. It means each pixel includes three primary color channels: red, green and blue. Various colors of the pixel can be produced by changing the value of those channels. Now, that makes the problem of finding the evidences of lane markings in RGB color space become extremely challenging. For example, to detect a specific color in RGB color space which is composed of three color channels, we must compare all three channels by turns. Therefore, in the case of white or yellow color of lane markings, the wide range in RGB color space of these two colors has to be covered completely.

(a) Gaussian smoothing ROI image with kernel size 9×9

(b) Grayscaling ROI image

Figure 4.4: Example of grayscaling step

Because of this reason, we will want to detect lane marking in grayscale image. Contrary to RGB image, grayscale image (also called black-white image) contains

only different shades of gray. It requires less information to be provided by specifying only one single intensity value for each pixel. Besides, a feature of lane markings is that it is always in a contrast color to road surface, for example, the most popular colors of lane markings are white and bright yellow in contrast to the dark gray color of the road. Therefore, the gray image can perfectly enhance this contrast and greatly help in improving the detection.

Before grayscaling, we try to smooth (also called blur) the ROI image a little by using the most popular smoothing method called "Gaussian Smoothing" or "Gaussian Filtering" with the kernel size 9×9. This step can help in intensifying the evidence of lane markings and eliminating unwanted noises from road surface. An example of smoothing or blurring image using OpenCV is in Figure 2.5. The result of the grayscaling step is illustrated in Figure 4.4.

Lane Feature Extraction

This thesis does not use some popular edge detection techniques such as Canny or Sobel edge detector but utilizes a fast and more effective technique proposed by Nieto et al. [23]. This technique is based on the assumption that, in a row of image, the pixels which belong to lane markings tend to have *"high intensity value surrounded by darker regions"* [23]. Thus, the detector independently filters each row of image by its pixels' intensity values.

We assume the values of pixels in a row of the image are $\{x_i\}_{i=1}^{w}$ (w: width of the image), then the filtered values $\{y_i\}_{i=1}^{w}$ are given by:

$$y_i = 2x_i - (x_{i-\tau} + x_{i+\tau}) - |x_{i-\tau} - x_{i+\tau}| \tag{4.1}$$

where τ is the parameter that approximates the width of lane marking in the image.

Figure 4.5: Grayscale ROI image after filtering

The filter highly responses to the pixels, which have higher intensity values than their left and right neighbors in the same row at distance τ (Figure 4.5). The last term of equation 4.1, $|x_{i-\tau} - x_{i+\tau}|$, is removed from filtered value y_i to help the filter less prone to errors, especially in the case that the difference between intensity values of left and right neighbors is too high.

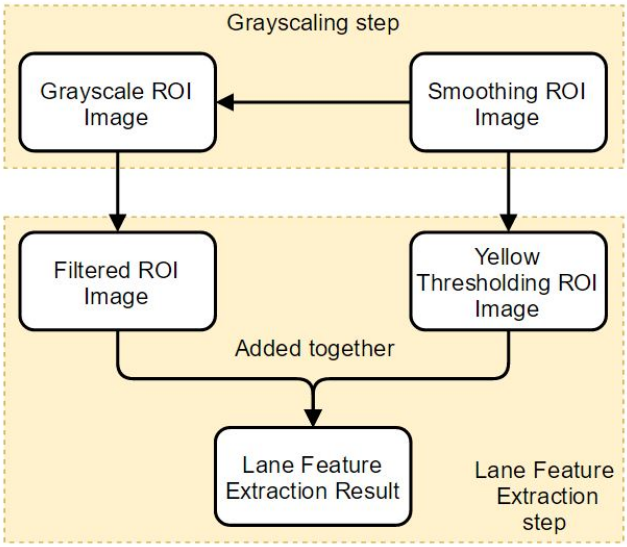


Figure 4.6: The detailed flow of lane feature extraction step

The disadvantage of the mentioned filter is that the filtering result for lane marking in yellow is not as good as in white, especially when the yellow color is not bright enough. Therefore, we apply an additional step called "yellow thresholding" to detect yellow color in RGB ROI image before grayscaling. The yellow thresholding image then is added with the filtered image to produce the result image of edge detection step. The detailed workflow is illustrated in Figure 4.6. Figure 4.7 demonstrates the result images in each step of edge detection in the case that there is a lane marking in yellow color.

(a) Road with lane marking in yellow

(b) Filtered image

(c) Yellow thresholding image

(d) Result image of lane feature extraction step

Figure 4.7: Example of lane feature extraction step

Thresholding

After many layers of Pre-processing stage, thresholding is the final one we use to decide which pixels we want to keep while discard the others based on their intensity values. In the image output from edge detection step (Figure 4.7(d)), the pixels that belong to lane-marking evidence tend to have higher intensity values. Besides, the disturbances may leave in the image many lower intensity pixels represented as thin and weak edges. Binary thresholding can help significantly in removing these noises and enhancing the "good" pixels. The basic idea behind binary thresholding is that all pixels that have intensity values higher than a certain threshold will be set to maximum value `maxVal`, while the others with intensity values below the threshold are set to zero [25]. The binary thresholding is given by the following formula:

$$\texttt{dst(x,y)} = \begin{cases} \texttt{maxVal} & \texttt{if src(x,y)} \geq \texttt{Threshold} \\ \texttt{0} & \texttt{otherwise} \end{cases} \tag{4.2}$$

where `dst(x,y)` and `src(x,y)` are respectively intensity values of the pixel in destination image and source image.

The result image after thresholding is demonstrated in Figure 4.8. This binary image is the final output of Pre-processing stage.

Figure 4.8: The final result of Pre-processing stage after thresholding

4.2.2 Lane Detection

Before diving into the details of Lane Detection stage, we should consider some characteristics of lane markings that can help in reducing a considerable amount of work and increasing detection accuracy. Those characteristics include:

- Left and right lane marking do not cross.
- Each lane marking only lies on a certain area.
- Lane markings should have a specific width.

The knowledge on lane marking characteristics may be exploited for lane detection. Initially, the binary image obtained from the Pre-processing stage (Figure 4.8) is split into two halves - left and right, each contains the evidences of one lane

marking (Figure 4.9). The lane detection procedure is then performed on each half image.

(a) Left half **(b)** Right half

Figure 4.9: Threshold image is split into two halves

As mention in section 4.1, Lane Detection is the stage that performs whenever the estimation of the lane markings is not available, therefore it consumes a lot of resource of the system. The stage is consisted of four consecutive steps as illustrated in Figure 4.10. First, hundreds of lines are randomly distributed in the binary half image. Each of them is then weighted with a score, which is calculated using an error function. Next, those lines are sorted based on their score (or weight). The best line, which has the highest score, will be the representation of lane marking. Besides, 50 other highest-score lines are also saved as candidates of lane marking and used as input for Lane Tracking stage.

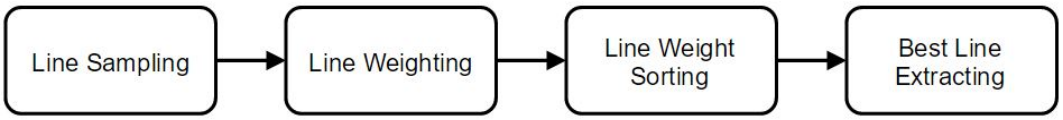


Figure 4.10: Lane Detection flow of Line-based Lane Detection Algorithm

Line Sampling

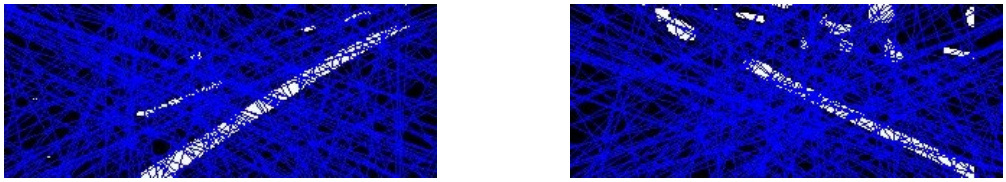
In this step, we try to distribute hundreds of lines over the binary half image. The more lines we use, the larger area in the image can be covered and the more chance we can find the correct lane marking.

Any lines can be define precisely by two different points. In this case, we only care about the area inside the binary half image, therefore, we choose the first point in the first row of the image - called start point with coordinate $sp(x_{sp}, 0)$ - and the second point in the last row of the image - called end point with coordinate $ep(x_{ep}, h)$, where h is the height of the image. Line position now is defined by

$$X = \{x_{sp}, x_{ep}\} \tag{4.3}$$

The work then is sampling x coordinates of each start point and end point. Assuming that the vehicle is running on the road, the lane markings are expected mostly inside

the two halves of ROI (Figure 4.9). Each set of x - x_{sp} and x_{ep} - is then chosen from a Gaussian (also called normal) distribution, which has the mean μ at the center of image row and the standard deviation σ equals to half of image's width that is large enough to cover the whole image, even for some cases start point or end point lies outside the image.



(a) Left half **(b)** Right half

Figure 4.11: Line sampling in two binary half images with 200 lines for each half

Figure 4.11 demonstrates line sampling in two halves of the ROI binary image. In this example, there are 200 lines distributed in each half, but in fact, to ensure a good detecting result, it requires using at least around 600 lines. Keep in mind that the number of lines used is the compromise between the accuracy and running speed. Larger number means better result but more calculation time.

Line Weighting

After line sampling, we try to weight all the lines by assigning each of them a score to measure the goodness. The score of a line is calculated as the number of white pixel in the binary image that the line and its neighbors contain. Here, because lane marking always has a certain width, the neighbors are included to increase the effectiveness of the score in measuring the goodness.

Figure 4.12 explains how to calculate the number of white pixels in a line. The slope s of the line is given by

$$s = \tan(\theta) = \frac{x_{ep} - x_{sp}}{h} \tag{4.4}$$

where θ is the angle between the line and image row, h is the height of the image, x_{sp} and x_{ep} are the x coordinates of start point and end point respectively.

To get the intensity value of each pixel of the line, we examine the line by each row of the image. Giving the current row of the image $y_{current}$, the column of the current point $x_{current}$ is defined by

$$x_{current} = x_{sp} + s \cdot y_{current} \tag{4.5}$$

where s is the slope of the line.

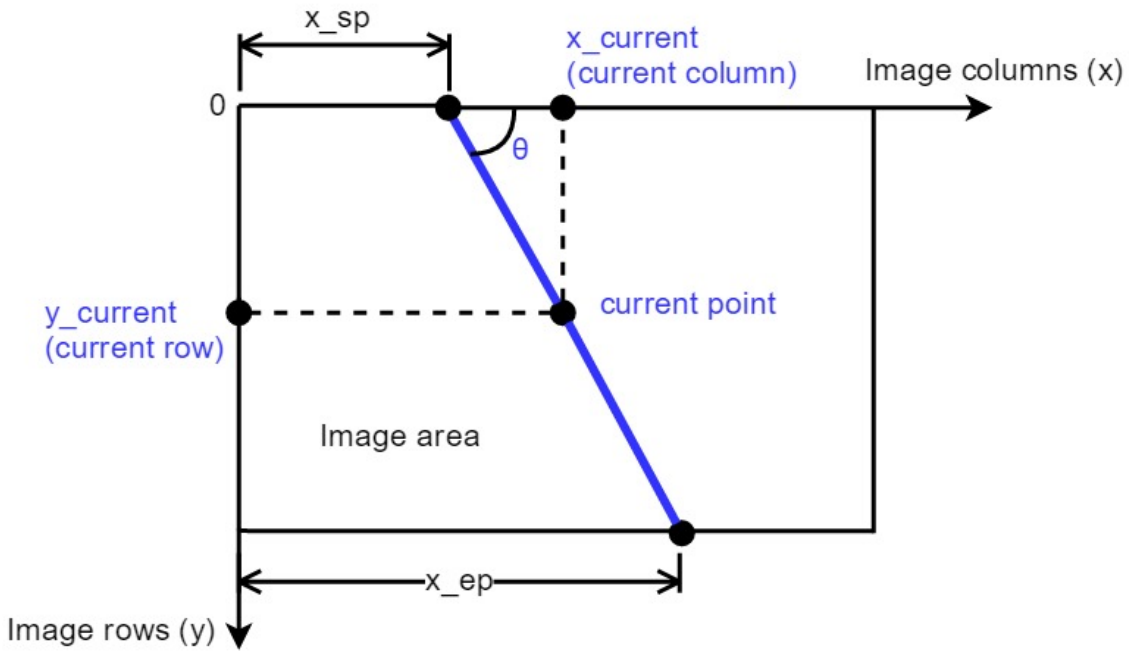


Figure 4.12: Line score calculation

By using fairly simple equations 4.4 and 4.5, we can easily calculate the score of each line distributed in the binary half image.

Line Weight Sorting and the Best Line Extracting

After finishing line weighting, all lines need to be sorted based on their scores. Thus, we have the lines in the order of how important they are. The most important line, which is the line with highest score, is the best representation of lane marking, as shown in Figure 4.13. Besides, not only the best line is kept, 50 lines that have the highest scores are stored as candidates of lane marking for being used later by Particle Filter in Lane Tracking stage. Note that the number of lines kept as candidates is adjustable, depending on how many particles the Particle Filter requires to work effectively.

(a) Left half **(b)** Right half

Figure 4.13: The best line is chosen as representation of lane marking

The result of Lane Detection stage in RGB image is demonstrated in Figure 4.14.

Figure 4.14: Lane detection result in RGB ROI image

Algorithm 2 shows the summary of Lane Detection stage in pseudo code.

Algorithm 2: Lane Detection stage in Line-based Lane Detection Algorithm

```
   // Calculating score of each line
 1 for i ← 0 to NUMBER_OF_ROLLS - 1 do
 2 |   y_start_point = 0
 3 |   y_end_point = image_height
 4 |   x_start_point = mean + Gaussian_sample
 5 |   x_end_point = mean + Gaussian_sample
 6 |   slope = (x_end_point − x_start_point) / image_height
 7 |   for j ← 0 to image_height - 1 do
 8 |   |   x_current = x_start_point + j · slope
 9 |   |   score += intensity_value_at(j, x_current)
10 |   end
11 end
   // Sorting lines
12 sort(vector_of_lines.begin(), vector_of_lines.end(), compare_score)
13 return line_with_highest_score
```

4.2.3 Lane Tracking

Lane Detection stage is slow and expensive to the system because it detects lane marking in current frame afresh without using any historical information. It is also easily distracted by noises and generates unstable results if running alone. Therefore, Lane Tracking stage is introduced to fix those problems. It is designed to be fast and to not consume too much resource of the system by using the information from the previous frame to estimate the position of lane marking in the current frame. Hence, in any cases, we will want to run it as much as we can. In fact, Lane Tracking stage performs whenever the estimated information of the previous frame is available.

In the first algorithm implemented in the thesis, we choose to use Particle Filter for Lane tracking stage because of its advantages as provided in Chapter 3. In some researches, Particle Filter can be utilized for both lane detection and lane tracking by running it for several iterations in detecting phase such as the algorithm described by Berriel et al. [4]. However, in this first algorithm, Particle Filter is

used for the only purpose of tracking the positions of lane markings in the current frame. The theory of Particle Filter has been presented in Section 2.4, and the detailed implementation of Particle Filter for lane tracking task will be discussed in this section.

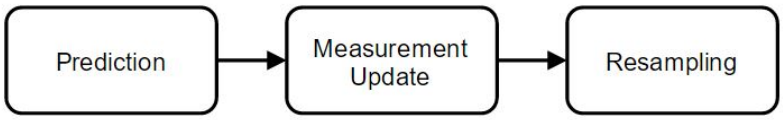


Figure 4.15: Lane Tracking flow

The Lane Tracking stage using Particle Filter includes three main steps in each iteration, which are prediction, measurement update and resampling, as shown in Figure 4.15. In Section 2.4, Particle Filter is introduced with four main steps including the initialization. However, in this case, we are allowed to skip the initialization step because there are already an input set of particles to represent the prior distribution of Particle Filter. This set is 50 "good" lines saved as candidates for lane marking from Lane Detection stage. Here, the number of particles used to track the position of lane marking in each side is adjustable. Again, it is the trade-off between the computation time and the accuracy of the result. Beside the set of "good" lines, the information from binary image provided by Pre-processing stage is used as measurement of Particle Filter in current frame.

Prediction

From the previous frame to the current frame, vehicle's position may change a little, therefore the lane marking should also move slightly to a new position. The particles need to follow this change to keep tracking lane marking. In Prediction step, filter tries to move each particle an appropriate distance based on its position in previous frame. Thus, the set of particles is still able to represent prior distribution in the new frame.

(a) Left half **(b)** Right half

Figure 4.16: Prediction step of Particle Filter in binary image

In our case, we adjust each particle by moving its start point and end point in the same row of the image. It means we add a random amount to the x coordinate of start point and end point. The random amount is sampled from a Gaussian

distribution with the mean is being the current value in x coordinate of start or end point and a small standard deviation $\sigma = 10/3$. Note that the standard deviation here is chosen empirically based on the width of lane marking in the binary image.

Measurement Update

This step is also called Weight Update. After Prediction step, we have all particles in new positions and they have the same weight. The work now is to assign a new weight to each of them to illustrate how important or how "good" it is in representing the lane marking with its new position.

First, to calculate the weight of a particle, we need to specify an error function. The error function should be capable of measuring how good a particle is by fitting it to the evidences of lane marking. Next, because each particle is a line with start point and end point as described in Lane Detection stage, then we can use the same technique presented in section 4.2.2 Line Weighting to calculate how many white pixels each line contains in the binary image and the line which has the largest number of white pixels should be the best representation of lane marking. Additionally, we realize that the maximum number of white pixels that a line can has is equal to the height of the image. Then the error $E(p)$ for a particle p can be given as

$$E(p) = h - score \tag{4.6}$$

where h is the image height and *score* is the number of white pixel that the line contains.

The error $E(p)$ implies exactly how good a particle is in representing lane marking. Now we calculate the weight of a particle based on its error. Keep in mind that for each particle, its weight needs to be inversely proportional to its error, as shown in the equation 4.7.

$$W_i^- = \frac{1}{1 + e^{E(p)}} \tag{4.7}$$

The weight of a particle is a probability, therefore we need to ensure that the total of all weights equals to 1 or $\Sigma W = 1$ by normalizing the weights we have received from equation 4.7.

$$W_i = \frac{W_i^-}{\sum W^-} \tag{4.8}$$

W_i is the new weights of particles in the current frame.

Resampling

This step is extremely important in helping the filter avoid the degeneracy problem and assure the convergence. The weights of particles calculated in the previous

step is used as sampling probability in Resampling. Thus, the particles with high weights have higher chance to be kept, while others which have low weights tend to be discarded.

In Line-based Lane Detection Algorithm, we use an algorithm called Low Variance Sampling described by Thrun et al. [28] for Resampling because of its easy implementation and robustness. This sampling algorithm is represent in pseudo code as shown in Algorithm 3. It uses a random number to sample from X, which is a set of M particles with corresponding weights W, and yet the probability of a particle to be sample is still proportional to its weight.

Algorithm 3: Low Variance Sampling [28]

```
1  X̄_t = ∅
2  r = rand(0; M^-1)
3  c = w_t^[1]
4  i = 1
5  for m ← 1 to M do
6  |   u = r + (m − 1) · M^-1
7  |   while u > c do
8  |   |   i ++
9  |   |   c += w_t^[i]
10 |   end
11 |   add x_t^[i] to X̄_t
12 end
13 return X̄_t
```

Figure 4.17 demonstrates the result after Lane Tracking stage. Based on the information of lane marking positions, we are now able to calculate a greatly important parameter which is lateral deviation of the car from the lane middle.

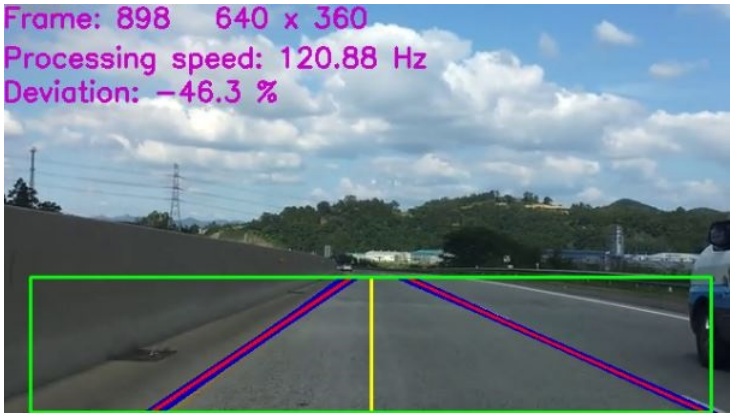


Figure 4.17: The final result of Line-based Lane Detection Algorithm

Lane Tracking stage is summarized by pseudo code in Algorithm 4.

Algorithm 4: Lane Tracking stage in Line-based Lane Detection Algorithm

```
1  X_t[NUMBER_OF_PARTICLES] // Set of particles
2  for i ← 0 to NUMBER_OF_PARTICLES - 1 do
3  |   y_start_point = 0
4  |   y_end_point = image_height
   |   // Prediction
5  |   x_start_point += Gaussian_sample
6  |   x_end_point += Gaussian_sample
   |   // Measurement update
7  |   slope = (x_end_point − x_start_point)/image_height
8  |   for j ← 0 to image_height - 1 do
9  |   |   x_current = x_start_point + j · slope
10 |   |   score += intensity_value_at(j, x_current) / 255
11 |   end
   |   // Weight update
12 |   weight[i] = 1 / (1 + exp(image_height − score))
13 end
   // Weight normalization
14 for i ← 0 to NUMBER_OF_PARTICLES - 1 do
15 |   weight[i] /= total_of_weights
16 end
   // Resampling
17 for i ← 0 to NUMBER_OF_PARTICLES - 1 do
18 |   draw i with probability ∝ weight[i]
19 |   add i to X_t
20 end
21 return X_t
```

4.2.4 Re-detection

Finally, in order to guarantee a good detecting and tracking result, several re-detection criteria are introduced. These criteria make sure that the best lines are good enough as representations of lane markings. They also check if the detected lane-marking positions are reasonable comply with the characteristics of lane markings as described in Section 4.2.2.

The re-detection criteria includes:

- Left and right lane marking do not cross.
- The distance between two lane markings muss be bigger than a threshold. This threshold may be changed in different roads.

- For each lane marking, at least 30% of it should lie inside the ROI. This criterion is useful when the vehicle is changing lane, both lane markings are shifting out of their ROI, system should re-detect lane markings in ROI instead of continuing tracking.
- The best line should have the score bigger than a threshold. This criterion is to avoid the situations in which Particle Filter tracks the wrong evidences or the number of evidences in the image is too small because the road temporarily has no lane markings.

In case at least one of these criteria is violated, the system will stop the Lane Tracking stage and trigger again the Lane Detection stage. Therefore, re-detection criteria help the system define when to perform lane detection, while run lane tracking all other time.

4.3 Spline-based Lane Detection Algorithm

In the second algorithm of the thesis, we implement a more complicated lane-position detection algorithm. The chosen approach uses more complex model that is cubic spline with four control points to represent lane markings. Particle Filter algorithm is still utilized for lane tracking with multi-dimensional particles corresponding to four control points of the cubic spline. Besides, the Inverse Perspective Mapping (IPM) technique is applied in Pre-processing stage to improve the result of the algorithm.

4.3.1 Pre-processing

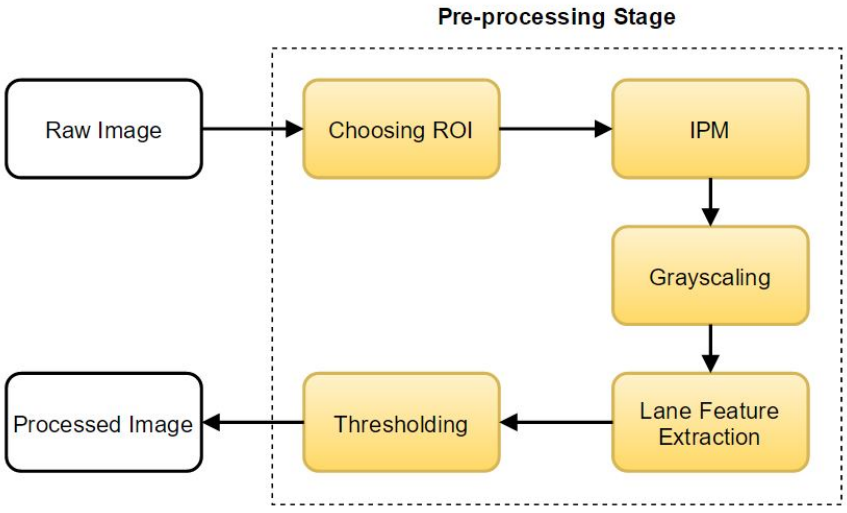


Figure 4.18: Flow of Pre-processing stage in the second algorithm

Basically, in this Spline-based Lane Detection Algorithm we use the same Preprocessing procedure as in Line-based Lane Detection Algorithm, the only difference is the IPM step is introduced after choosing ROI (Figure 4.18). Therefore, this section only discusses the concepts of IPM technique and how to use it in our algorithm.

Inverse Perspective Mapping (IPM)

Inverse Perspective Mapping is a technique that transforms the original perspective image captured from camera to a bird's-eye-view image, therefore it is sometimes called bird's-eye-view mapping. The benefit of this step is that it removes the perspective effect in the original image and so that lane markings in IPM image now appear to be vertical and parallel. The transformation then allows us to work with a constant lane width in the IPM image and provides a mapping from original image pixels to real-world distances.

To realize the IPM transformation, first we assume that the road is flat. Next, we need to define the 3×3 homography matrix $\mathbf{H}$ that maps the point (u, v) in the original image to the corresponding point (x, y) in the IPM image.

$$\begin{pmatrix} \lambda x \\ \lambda y \\ \lambda \end{pmatrix} = \mathbf{H} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{4.9}$$

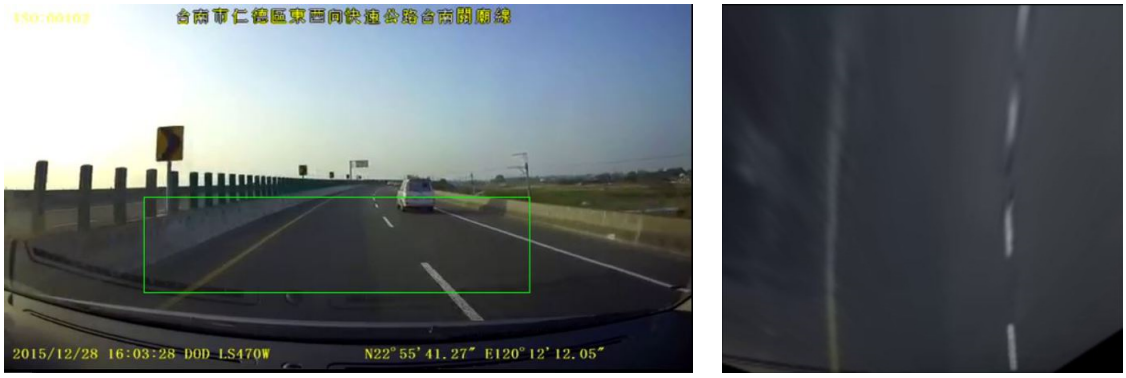
There are two ways to obtain matrix $\mathbf{H}$:

- The first method, which was proposed by Bertozzi and Broggi [5], is based on the position and angle of camera in the real-world coordinate system. This method uses many complicated calculations but has an advantage of having no required measurement in real world.
- The second way, which is described in [14], is much simpler and more popular. It uses four reference points to externally calibrate the camera. To get the best result, this method requires the correct distances of four corresponding points in real world. By solving the equation 4.9 for four couples of points, we can easily obtain matrix $\mathbf{H}$.

For both methods, matrix $\mathbf{H}$ only needs to be calculated once and then it can be reused to transform all other images. In the case that we want to transform the IPM image back to the perspective image, simply use $\mathbf{H}^{-1}$.

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathbf{H}^{-1} \begin{pmatrix} \lambda x \\ \lambda y \\ \lambda \end{pmatrix} \tag{4.10}$$

In this thesis, we choose to use the second method to find matrix **H**. Four reference points are selected manually in the original image. Figure 4.19 and 4.20 demonstrate several results in Pre-processing stage of Spline-based Lane Detection Algorithm.



(a) Original image with the chosen ROI **(b)** The IPM image

Figure 4.19: Applying IPM transformation to the ROI image

(a) Left half **(b)** Right half

Figure 4.20: Result of Pre-processing stage

4.3.2 Lane Detection

Lane Detection stage of Spline-based Lane Detection Algorithm is performed in the IPM binary images output from Pre-processing stage (Figure 4.20). Keep in mind that, this time we use cubic spline to represent lane markings and it is a much more complex model in compare with straight line. The theory about the cubic spline is presented in Section 2.5.

In general, the Lane Detection stage of Spline-based Lane Detection Algorithm follows the same procedure as the one of Line-based Lane Detection Algorithm (Figure 4.21). However, because of the complexity of cubic spline model, we should use additionally Particle Filter to support detection process.

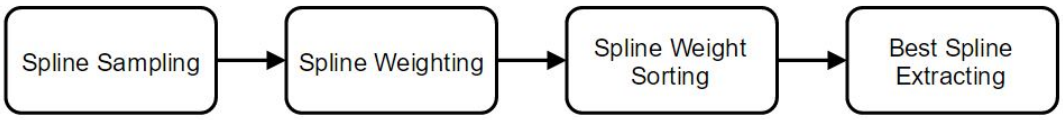


Figure 4.21: Lane Detection flow of Spline-based Lane Detection Algorithm

Spline Sampling

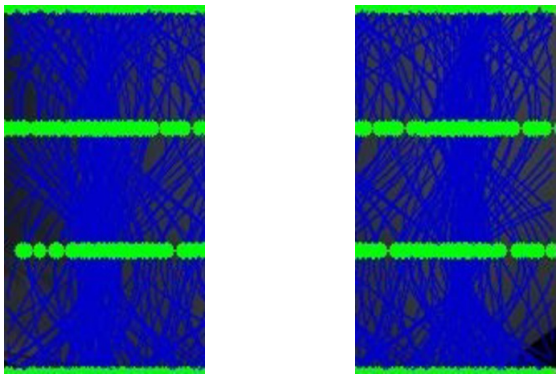
By using cubic spline model, the first thing we need to care about is how many number of control points we should use and where to place them. Kim [17] in his research tests the usage of two, three and four control points and points out that using four control points is not only better in representation but also can maintain the robustness of the system. The nearest control point is put at the bottom of IPM image, while the furthest one is at the image's top row. Two other points are chosen so that they lie between two first points and the spacing between them is as equal as possible. To make it simple, in this thesis, control points are chosen as following: assume we use N control points $c_i = (x_i, y_i)$ where x_i and y_i are the coordinates of control points in the image, then the y_i value of each control points is given by

$$y_i = i \cdot \left(\frac{h}{N-1}\right) \tag{4.11}$$

where i is an integer and $0 \leq i \leq N-1$, h is the height of IPM image. Now the cubic spline is defined by the x coordinates of control points:

$$X = \{x_1, x_2, \cdots, x_N\} \tag{4.12}$$

Now, the situation becomes the same as in Line-based Lane Detection Algorithm. Each x_i will be sampled from a Gaussian distribution, which has the mean μ at the center of IPM image row and the standard deviation σ equals to half of image's width.



(a) Left half **(b)** Right half

Figure 4.22: Spline sampling in two binary half images with 100 splines for each half

Figure 4.22 shows an example of sampling 100 splines in each half of the IPM binary image. In fact, because of the complexity of cubic spline model, the number should be thousands to ensure a good detecting result . A different way using less number of splines is running few iterations of Particle Filter in Lane Detection stage. This method could be slightly slower but very useful in case lane markings are in unusual positions.

Spline Weighting

Similar to Line Weighting in Line-based Lane Detection Algorithm, now we try to assign a weight for each spline to describe how good it is in representing lane marking. First, we define the equation of each cubic spline, after that we are able to calculate its score which is the number of white pixels it and its neighbors contain. Unlike the simplicity of line equation, cubic spline equation is the combination of three cubic polynomials. The process of defining these polynomials is called spline interpolation that is described in Section 2.5. After calculating equation of the spline, we can easily obtain its score by examining each pixel of it in the binary image.

Spline Weight Sorting and the Best Spline Extracting

The final step in the Lane Detection stage is sorting the splines we have distributed in the IPM image based on their scores, then choosing the one with highest score as the final result of detecting stage to represent lane marking. Similar to the first algorithm of this thesis, a number of splines which have highest scores are kept as candidates of lane marking and will be used as input for Particle Filter in Lane Tracking stage.

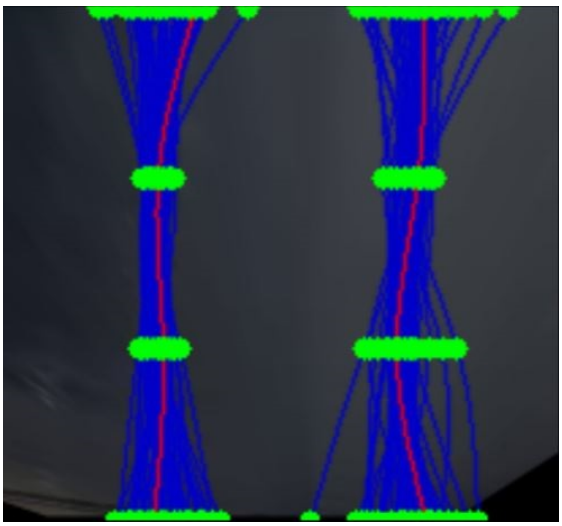


Figure 4.23: 50 splines with highest scores (in blue) each side are kept for next stage, the best line (in red) is the result of Lane Detecting stage

Lane Detection stage of Spline-based Lane Detection Algorithm is summarized as following:

Algorithm 5: Lane Detection stage in Spline-based Lane Detection Algorithm

```
   // Calculating score of each spline
 1 for i ← 0 to NUMBER_OF_ROLLS - 1 do
 2 |   control_point1 = (mean + Gaussian_sample, 0)
 3 |   control_point2 = (mean + Gaussian_sample, image_height / 3)
 4 |   control_point3 = (mean + Gaussian_sample, 2 · image_height / 3)
 5 |   control_point4 = (mean + Gaussian_sample, image_height)
 6 |   spline_interpolation(control_point1, control_point2, control_point3,
 7 |   control_point4)
 8 |   for j ← 0 to image_height - 1 do
 9 |   |   score += intensity_value_at(j, x_current)
10 |   end
11 end
   // Sorting splines
12 sort(vector_of_lines.begin(), vector_of_lines.end(), compare_score)
13 return spline_with_highest_score
```

4.3.3 Lane Tracking

Spline-based Lane Detection Algorithm also uses Particle Filter for Lane Tracking stage with a very similar flow as described in Section 4.2.3 (Figure 4.15). The main difference is the stage performs in IPM binary image using cubic splines as particles. The set of "good" splines saved in Lane Detection stage is used as candidates of lane-marking representation in this stage. These candidates can be considered to be inputs of the Particle Filter that represent prior distribution. The steps of Particle Filter is described as following:

- **Prediction**: in this step, the new particles, here are cubic splines, are generated by moving the control points of the old particles from previous frame. Thus, the x coordinate of each control point is added an amount which is randomly sampled from a Gaussian distribution with the mean being the current x value and a small standard deviation. The standard deviation is different for each control point of a spline. The further control point tends to move a lot, therefore it requires a bigger standard deviation, while the closest one mostly stands still.

- **Measurement Update**: in this step, the new particles will be scored based of their goodness in representing lane marking. The technique is the same as described in Section 4.3.2. That means we also need to define the equations of the new particles using cubic spline interpolation (Section 2.5). After obtaining

the score of each particle, we can calculate its weight by using the set of equations 4.6, 4.7 and 4.8.

- **Resampling**: Finally, the set of particles is re-sampled to avoid degeneracy problem and guarantee the convergence of the filter. The return is a new set of particles in which the particles with low weight have been discarded and replaced by better ones. In the new set of particles, the one with the highest score will be chosen as the final result (Figure 4.24). This step exploits the algorithm Low Variance Sampling as described in Algorithm 3.

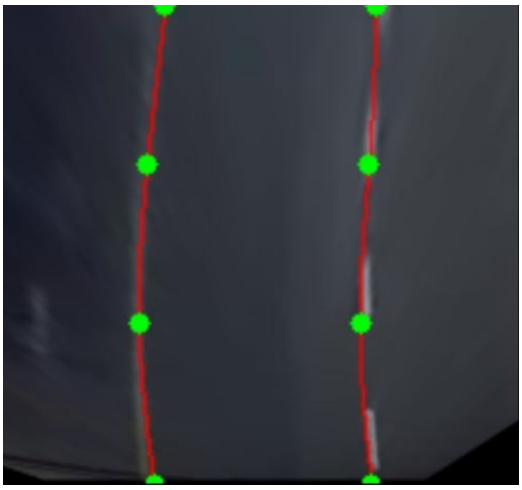


Figure 4.24: The result of Lane Tracking stage

After obtaining the results in IPM image, we need to convert them back to perspective space of the original image to receive the final result. Here, we discard the furthest control point because of its nature to be unstable and secure the most accurate and stable result.

Figure 4.25: The final result of Spline-based Lane Detection Algorithm

4.4 Hough-based Lane Detection Algorithm

In the third algorithm of this thesis, we choose a totally different approach which models lane markings by straight lines and uses Hough Transform integrated with Kalman Filter in lane-marking detection and tracking. This approach can help lane detection system achieve higher speed with an adequate accuracy in common scenarios. Deriving from general workflow (Figure 4.1), the more specific flow of Hough-based Lane Detection Algorithm is shown in Figure 4.26.

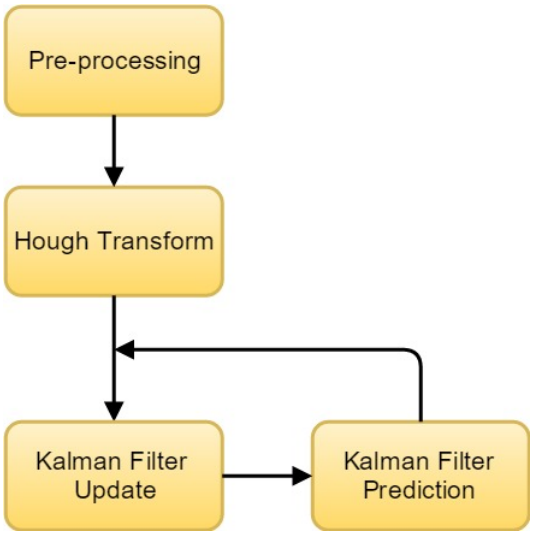


Figure 4.26: Workflow of Hough-based Lane Detection Algorithm

Hough-based Lane Detection Algorithm uses the same Pre-processing stage as Line-based Lane Detection Algorithm (4.2.1), therefore, in this section, we only discuss Lane Detection and Lane Tracking stage.

4.4.1 Lane Detection

The Lane Detection stage of this third algorithm is realized using Hough Transform and is performed on ROI binary image outputs from Pre-processing stage (Figure 4.8). The theory of Hough Transform is presented in Section 2.6.

Unlike lane detection technique using in Line-based Lane Detection Algorithm (Section 4.2.2), lines detection method using Hough Transform does not require splitting binary image into two halves. Hough Transform executes pretty fast in the whole image, and after the process, the output is a set of lines in the image which are expressed in polar form:

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{4.13}$$

where with θ is the angle between the line and x axis of the image and ρ is the distance from the image origin to the line (see Figure 4.27).

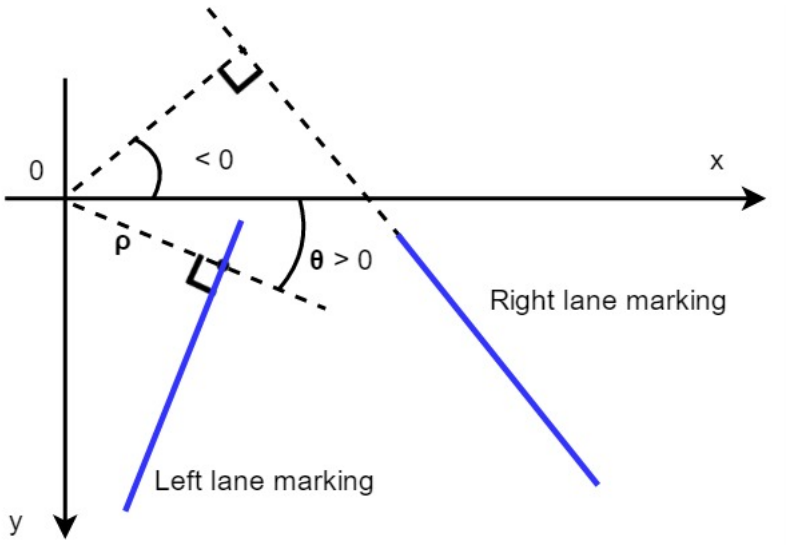


Figure 4.27: Lines expressed in polar form (ρ, θ)

The threshold used in Hough Transform is extremely important, from definition, it is *"the minimum number of intersections to "detect" a line"* [24]. The threshold is chosen based on the dimensions of image in which Hough Transform performs, a larger image should have a higher threshold. A suitable threshold can help a lot in discarding wrong lines.

Beside choosing the threshold, we can do several other refinements on the output set of lines from Hough Transform based on characteristics of lane markings (as presented in Section 4.2.2):

- A maximum value of θ is proposed to remove the lines which have unreasonable positions for lane markings.
- Lines which have $\theta \geq 0$ are considered as candidates of left lane marking, while ones with $\theta < 0$ are candidates of right lane markings.

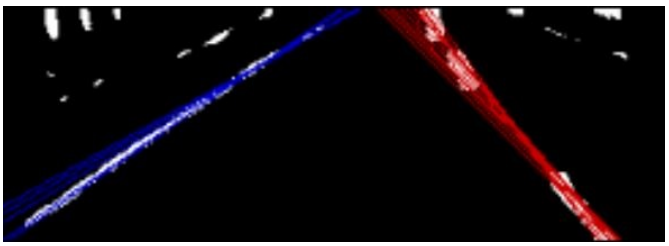


Figure 4.28: Output of Hough Transform in ROI binary image: left lane-marking candidates in blue, right lane-marking candidates in red

In each line outputted from Hough Transform, only the line segment inside ROI area is our concern. Therefore we take one more step to fit these line inside ROI. This step only needs a few basic operations with line equation, but it is significantly useful in helping calculate the score of each line. The same technique used in Section 4.2.2 is applied here to compute line scores. Lines from set of left and right lane-marking candidates will be compared separately. Two lines which have highest scores of each set are chosen as final output of Lane Detection stage.

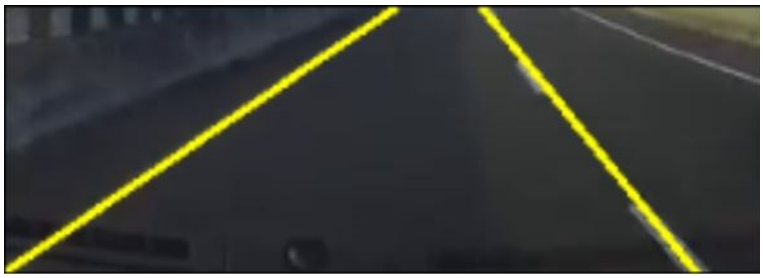


Figure 4.29: The best Hough line each side is chosen as final output of Lane Detection stage

4.4.2 Lane Tracking

Lane Tracking stage of Hough-based Lane Detection Algorithm is made by applying Kalman Filter. Working under Markov assumptions, Kalman Filter can provide a more time-effective solution to the system. The theory of Kalman Filter is presented in Section 2.3.

Lane Tracking stage using Kalman Filter can be divided into two big steps: Kalman Filter Update and Kalman Filter Prediction (see Figure 4.26). The measurement used to update the filter each frame is the best line each side outputs from Lane Detection stage (Figure 4.29). The details of each steps and the set of equations used are the same as described in Section 2.3. In this specific situation, to predict parameters of best lines in polar form (ρ, θ) in both sides using Kalman Filter, the state vector $x(t)$ and observation vector $z(t)$ are defined by

$$x(t) = z(t) = \begin{bmatrix} \rho_l(t) & \dot{\rho}_l(t) & \theta_l(t) & \dot{\theta}_l(t) & \rho_r(t) & \dot{\rho}_r(t) & \theta_r(t) & \dot{\theta}_r(t) \end{bmatrix}^T \tag{4.14}$$

where ρ_l and θ_l are position parameters of left-side line in polar form, $\dot{\rho}_l$ and $\dot{\theta}_l$ are the derivatives of ρ_l and θ_l which are approximated by the differences between two consecutive time frames. ρ_r, $\dot{\rho}_r$, θ_r and $\dot{\theta}_r$ are similar parameters to track right-side line.

The state transition matrix $\mathbf{A}$ should be a 8×8 matrix:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.15}$$

and the matrix $\mathbf{H}$ which associates the state x to the measurement z is the identity matrix. Additionally, keep in mind Kalman Filter performs under the assumption that noises of the process and and measurement are "white" or not correlated in time and with each other. Therefore, we can make the noise covariance matrices of the process and measurement to be diagonal and constant.

The final result of Hough-based Lane Detection Algorithm is demonstrated as Figure 4.30.

Figure 4.30: The final result of Hough-based Lane Detection Algorithm

4.5 Conclusion

In this chapter, we present in-detail concepts of three different approaches we have chosen to implement in this thesis. These choices are based on the in-depth research on the prior work on lane detection systems as presented in Chapter 3. We also take into account the requirements of this thesis for a lane detection system that is able to work in real-time based on a low power embedded hardware. The methods used in the thesis are the trade-off between the computation time, so that

the system can run at a speed at least 20 FPS, and an acceptable accuracy of final result in common working scenarios.

All three approaches in the thesis follow the general flow as shown in Figure 4.1 with three main stages: Pre-processing, Lane Detection and Lane Tracking. Pre-processing stage includes several low-level image processing steps. Its output is a binary image containing white pixels which are the indicators of lane markings. Lane Detection and Lane Tracking stage are performed on the binary image outputs from Pre-processing stage using the information of lane-marking model. While Lane Detection stage tries to estimate the position of lane marking without knowledge from previous frames, Lane Tracking stage uses prior estimation to predict the position of lane marking in current frame. Therefore, in compare to Lane Detection, Lane Tracking process is much faster and inexpensive to the system.

In the first approach, Line-based Lane Detection Algorithm, the lane markings are modeled by simple straight lines. Lane Detection stage is realized by sampling hundreds of lines over the whole area of the binary image and finding the best line that matches the evidence of lane marking. Finally, a Particle Filter is utilized for Lane Tracking stage. The second approach, Spline-based Lane Detection Algorithm, is the most complicated one in all three algorithms. It uses IPM technique in Pre-processing stage to remove perspective effect in the original image, the algorithm then works on IPM image. Lane marking in Spline-based Algorithm is modeled by a more complex model - cubic spline with four control points. Particle Filter is still the choice for Lane Tracking stage of this algorithm. The last approach of the thesis, Hough-based Lane Detection Algorithm, focuses on improving the speed of the system with an adequate accuracy. It uses a method called Hough Transform to find the candidate lines of lane-marking representation, the one with highest score will be chosen as the best measurement. In Lane Tracking stage, a Kalman Filter will use this measurement to estimate the final position of lane marking.

The next chapter will discuss the in-depth implementation in real hardware of all three approaches.

5 Implementation

This chapter discusses the details of system structure and its relation to other components. The in-depth implementation of all three lane-position detection algorithms in the thesis are also presented.

5.1 General Structure of the System

The final goal of the thesis is an lane-position detection algorithm running on our available embedded hardware Raspberry Pi 3. The detail of this hardware is presented in Chapter 6. Illustration of the general system structure is shown in Figure 5.1.

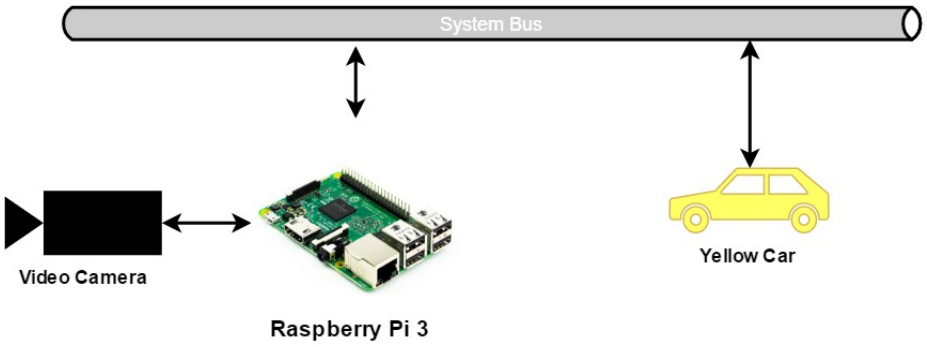


Figure 5.1: General system structure

First, a monocular video camera is used to provide the input frames of image for the system. The price of a device like camera has been enormously reduced recently, but it is recommended to use an above-average video camera which is able to provide enough details of lane markings in a good contrast image. Our algorithm has been tested to work with input video at quality $240p$ and frame ratio $16:9$, but we recommend to use at least $360p$ input video for the system to guarantee the best detecting result.

Next, the input data is processed by the lane-position detection program implemented on Raspberry Pi 3, an extremely popular embedded platform. The output which is the relative position of the vehicle with respect to lane markings will be sent through a Controller Area Network (CAN) interface to the system CAN bus and may be used by other systems such as Automated Vehicle Control System.

5.2 General Component Diagram

The general component diagram illustrates the implementation of general workflow as described in the Section 4.1. The details of each component will be discussed in following sections.

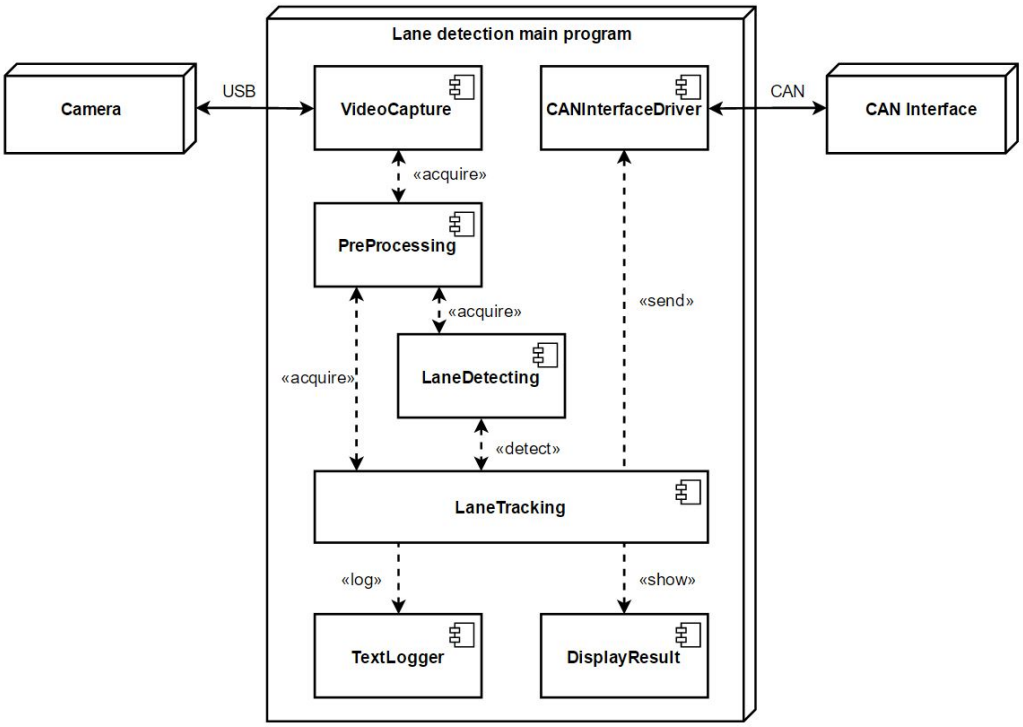


Figure 5.2: General component diagram

Components	Description
`VideoCapture`	The component to control the input video stream
`PreProcessing`	The image processing module using OpenCV
`LaneDetecting`	Lane detecting in the processed images
`LaneTracking`	Lane tracking with Particle Filter or Kalman Filter
`TextLogger`	Output data in text files for later evaluation
`DisplayResult`	Display detected lanes as overlay on output video
`CANInterfaceDriver`	The driver to communicate with CAN interface

Table 5.1: Components of implemented main program

5.3 Descriptions of the Components

This section describes in detail each component of the implemented main program. Except `LaneDetecting` and `LaneTracking`, the descriptions of other components are shared by all three thesis algorithms.

VideoCapture

This component includes several functions to control input video stream, for example start, stop, count number of frames, etc.

Return	Functions	Description
constructor	`VideoCapture(const string& filename)`	Capture video stream from a file
constructor	`VideoCapture(int device)`	Capture video stream from a device
`bool`	`isOpened() const`	Check if the video stream is opened successfully
`bool`	`read(cv::Mat& image)`	Read a frame from the video stream
`double`	`get(int propId)`	Get the value of the property with the ID number `propId`
`bool`	`set(int propId, double value)`	Set the value `value` to the property with the ID number `propId`

Table 5.2: Functions called in `VideoCapture` component

PreProcessing

`PreProcessing` includes several low-level image processing functions such as grayscaling, blurring, thresholding, etc. Its parameters are described in Table 5.3.

Parameters	Description
`src`	Source image
`dst`	Destination image
`laneDetectorConf`	Configuration of a particular dataset

Table 5.3: Parameters of `PreProcessing`

Functions used in `PreProcessing` component are given as below

Return	Functions	Description
`void`	`GaussianBlur(cv::Mat& src, cv::Mat& dst, Size ksize, double sigmaX, double sigmaY, int borderType)`	Apply Gaussian blur to the `src` image, the result is stored in `dst`
`void`	`cvtColor(cv::Mat& src, cv::Mat& dst, int code, int dstCn)`	Convert image from one color space to another, `code = CV_BGR2GRAY` or `CV_RGB2HSV`
`void`	`inRange(cv::Mat& src, cv::Scalar& lowerb, cv::Scalar& upperb, cv::Mat& dst)`	Mask elements of `src` that do not lie between the elements of lower boundary `lowerb` and upper boundary `upperb`
`void`	`FilterImage(cv::Mat& src, cv::Mat& dst, int tau)`	Filter the `src` image by the filter described in Section 4.2.1
`void`	`addWeighted(cv::Mat& src1, double alpha, cv::Mat& src2, double beta, double gamma, cv::Mat& dst, int dtype)`	Compute weighted sum of two arrays `dst = alpha · src1 + beta · src2 + gamma`
`void`	`threshold(cv::Mat& src, cv::Mat& dst, double thresh, double maxval, int type)`	Apply a fixed threshold to the `src` image

Table 5.4: Functions called in `PreProcessing` component

TextLogger

`TextLogger` is a simple module of the main program. It helps in exporting output data to text files, this data then can be used for testing and evaluating the lane-detection algorithm. The output data includes left and right lane positions, lane width and frame number. Figure 5.3 illustrates the structure of an output text file, the first column includes frame numbers, the second and third column are left and right lane-marking positions with respect to the frame number, and finally, the lane

width is calculated from the positions of left and right lane marking in the last column.

```
0000   34.1292   477.257   443.127
0001   32.8303   480.821   447.991
0002   31.5314   479.522   447.991
0003   30.2325   478.223   447.991
0004   28.9336   476.924   447.991
0005   27.6346   475.625   447.991
0006   28.2227   476.213   447.991
0007   28.8107   474.914   446.104
0008   29.3988   475.502   446.104
0009   28.0999   476.090   447.991
0010   26.8010   486.080   459.279
0011   30.3652   484.781   454.416
0012   40.3549   485.369   445.014
```

Figure 5.3: Example of the structure of output text file

DisplayResult

`DisplayResult` is the component that is responsible for drawing lane-position detection results onto the image frame as overlay by using several built-in drawing functions of OpenCV, for example `circle`, `line`, `rectangle`, `fillPoly`, etc. The information drawn includes left and right lane markings, frame number, frame dimensions, time required to process each frame and lateral deviation of the vehicle (see Figure 4.17, 4.25 and 4.30).

CANInterfaceDriver

This is the driver to control the communication with CAN interface hardware (Tiny-CAN II-XL). The output of lane detection program will be sent to the CAN system bus as messages and used by other systems. For example, the most important parameter that the lane-position detection system sends through CAN bus is the distance between the vehicle and lane center. This parameter is calculated as an integer to measure the distance in *centimeter*. The number is converted to hexadecimal and then divided into the high 8 bits and the low 8 bits to send through CAN bus. For instance, the number 300 is represented in hexadecimal as **012C**. The low 8 bits are **2C** and the high 8 bits are **01** (see Figure 5.4).

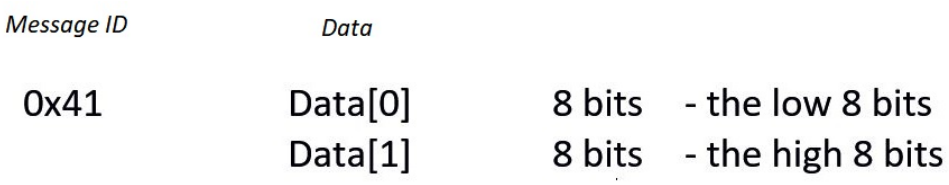


Figure 5.4: Example of the structure of CAN message

5.3.1 Line-based and Spline-based Lane Detection Algorithm

`LaneDetecting` and `LaneTracking` of Line-based and Spline-based Algorithm have the same description as the following.

LaneDetecting

`LaneDetecting` implements the flow described in Figure 4.10. Its parameters are given as below

Parameters	Description
`img`	The image in which the lane marking is being detected
`candidates`	Best lines are saved as `candidates` of lane marking for the next stage

Table 5.5: Parameters of `LaneDetecting` of Line-based and Spline-based Lane Detection Algorithm

LaneTracking

The `LaneTracking` of Line-based and Spline-based Algorithm is implemented using Particle Filter. The name of the function is `Particle_Filter()`, and its parameters are described as in Table 5.6.

Parameters	Description
`img`	The image in which the lane tracking is performing
`particles`	The particles of the filter which uses candidates from `LaneDetecting` as arguments
`maxWeight`	The maximum score of the all particles

Table 5.6: Parameters of `LaneTracking` of Line-based and Spline-based Lane Detection Algorithm

5.3.2 Hough-based Lane Detection Algorithm

LaneDetecting

The `LaneDetecting` of Hough-based Algorithm is implemented using Hough Transform. Its parameters are given in Table 5.7.

Parameters	Description
`img`	The image in which the lane detecting is performing
`laneDetectorConf`	Configuration of a particular dataset
`hfLanes`	The detected lane markings in form of Hough lines (polar form)
`imgRGB`	The image `img` in RGB color space in which results of Hough Transform are drawn

Table 5.7: Parameters of `LaneDetecting` of Hough-based Lane Detection Algorithm

Functions called in `LaneDetecting` of Hough-based Algorithm are given in Table 5.8.

Return	Functions	Description
`void`	`HoughLines(cv::Mat& image, std::vector<cv::Vec2f> lines, double rho, double theta, int threshold, double srn=0, double stn=0)`	Apply Hough Transform to `image`, results are saved in a vector of `lines`
`void`	`HfLanetoLane(const cv::Mat& laneMat, const std::vector<cv::Vec2f>& hfLanes, std::vector<Lane>& lanes)`	Convert lane lines from Hough form to normal form defined by two points
`void`	`CalculateLanePoints(const cv::Mat& img, Lane& lane)`	Fit lane lines inside ROI image `img`
`void`	`CalculateLaneScore(const cv::Mat& img, Lane& lane)`	Calculate the score of each line detected by Hough Transform

Table 5.8: Functions called in `LaneDetecting` of Hough-based Lane Detection Algorithm

LaneTracking

The `LaneTracking` of Hough-based Algorithm is implemented using Kalman Filter. The name of the function is `Kalman_Filter()`, and its parameters are described as in Table 5.9.

Parameters	Description
`img`	The image in which the lane tracking is performing
`laneKalmanFilter`	The Kalman Filter object of the OpenCV class `cv::KalmanFilter`
`laneKalmanMeasureMat`	The measurement matrix of the filter
`hfLanes`	The measurement in current time frame
`lastHfLanes`	The tracking result of the last time frame
`preHfLanes`	The predict result of current time frame
`postHfLanes`	The final result after update with the measurement

Table 5.9: Parameters of `LaneTracking` of Hough-based Lane Detection Algorithm

5.4 Applied Software

The software and library used in the thesis are introduced in this section.

5.4.1 C++ and OpenCV Library

As introduced in Section 2.2, OpenCV is a C++ open-source library for image processing and computer vision. Originally developed by Intel, the library is considered the most efficient way to manipulate images. Therefore, in this thesis, we have chosen OpenCV library to realize all low-level image processing steps such as reading image frame, blurring, grayscaling, thresholding, etc. The stable version of OpenCV used in the thesis is 2.4.13.0. Homepage of OpenCV library can be visited at the address *http://opencv.org/*, and its inclusive documents can be found online at *http://docs.opencv.org/2.4/modules/refman.html* and *http://docs.opencv.org/2.4/doc/tutorials/tutorials.html*. An great book that has

helped us a lot in using OpenCV and image processing is of Bradski and Kaehler [8].

All software in the thesis is written in C++ programming language to exploit its efficiency and convenience with OpenCV library.

5.4.2 Visual Studio Community 2015

The software of the thesis is developed using Visual Studio Community 2015, a popular Integrated Development Environment (IDE) from Microsoft. Announced from 2014, Visual Studio Community is not only a new free version of Visual Studio, but also fully-featured IDE for students, open-source and individual developers. Its newest version can be downloaded from *https://www.visualstudio.com/vs/community/*.

5.4.3 CMake

CMake is an open-source and platform-independent build tool which is designed to build, test and package software. It helps in creating appropriate project files from source files based on a specific template for the users' desired platform. In the thesis, CMake is utilized to help in building OpenCV library for Windows environment before the library can be used in Visual Studio projects. Besides, the project files of each lane-detection algorithm in Raspbian environment of Raspberry Pi 3 are also created with CMake. The CMake homepage can be visited at *https://cmake.org/*. The version used in the thesis is 3.6.2.

5.5 Conclusion

In this chapter, we present thoroughly the implementation of the system in the thesis. The general system structure includes a video camera that provides input video stream to the lane-position detection program running on Raspberry Pi 3. The output data can be sent to other systems as messages through a system CAN bus.

The realization of lane-position detection program implemented on Raspberry Pi 3 is also discussed. The software is developed using C++ programming language with the support of OpenCV library. The main program includes several main components such as `PreProcessing`, `LaneDetecting`, `LaneTracking` and `DisplayResult`. The chapter also presents the detailed descriptions of each component with parameters and functions called.

The next chapter covers testing results and evaluations of each algorithm with several given datasets.

6 Testing and Evaluation

The final step after the implementation of all three lane detection algorithms is testing and evaluating. This chapter presents the steps of testing procedure, along with the testing results of each algorithm in the given datasets. Based on these results, each algorithm is then analyzed and evaluated to consider its capability to satisfy the initial requirements of this thesis.

6.1 Datasets and System Test-Bed Configuration

This section presents the details of the used datasets and the system setup for testing and evaluation.

6.1.1 Datasets

Our algorithms are tested with different datasets from many sources, all of them are published on the Internet so that people are free to access. Each dataset includes from hundreds to a few thousands of frames and was recorded in different scenarios such as highways or urban streets. The details of each algorithm are presented in Table 6.1. The first dataset is taken from KITTI[1] datasets [11] (homepage *http://www.cvlibs.net/datasets/kitti/*) which is captured in the streets of the city of Karlsruhe, Germany; the next one is recorded in a highway in Taiwan and last one is at night in a highway of Toronto, Canada.

Name of the Dataset	**Type**	**Light Condition**	**Number of Frames**	**Dimensions**
KITTI	Urban street	Day	339	1242×375
HwTaiwan	Highway	Day	3000	640×360
HwNight	Highway	Night	5000	640×360

Table 6.1: Datasets used for testing and evaluating

[1]Karlsruhe Institute of Technology (KIT) and Toyota Technological Institute at Chicago (TTI-C)

Most of situations in these datasets are quite common in daily-life driving with normal traffic intensity. There are only some challenging scenarios with critical light intensity or when lane markings are faded or covered by other vehicles. Some images drawn from the datasets are shown in Figure 6.1.

(a) KITTI

(b) HwTaiwan

(c) HwNight

Figure 6.1: Some images from four datasets

6.1.2 Setup

The setup of the system for testing and evaluation is as shown in Figure 6.2. First, a PiCAN CAN-bus board is plugged to the GPIO pins of Raspberry Pi 3. PiCAN CAN-bus board is a product of SK Pang Electronics (homepage *http://skpang.co.uk/*), which provides CAN-bus capability for Raspberry Pi. Next, the PiCAN CAN-bus board is connected to a CAN interface hardware - Tiny-CAN II-XL - which is a CAN-USB adapter from MHS Elektronik (homepage *http://www.mhs-elektronik.de/*). This CAN interface is then connected to the system CAN bus and helps in sending the messages output from Raspberry Pi 3 to the other systems.

For testing purpose, the datasets are stored in the microSD card of Raspberry Pi 3 and used by the lane-position detection program. These datasets have been already processed to reduce the resolution to the quality as presented in the section 6.1.1. However, in reality, Raspberry Pi 3 uses input video stream from a connected camera and the process to reduce the resolution of the video captured by the camera can cost a lot of system resource. Therefore, if it requires to adjust the resolution of input video, we recommend to perform this process in another CPU before providing

the video to Raspberry Pi 3, so that the working speed of lane-position detection may be assured to be the same as we present here.

Besides, to compare the working speed of the system between using a real camera and using datasets from its microSD card, we also connect the system to a Raspberry Pi Camera Board v1.3 which is a 5-*Megapixels* camera module for Raspberry Pi (the details about this camera module can be found at *https://www.raspberrypi.org/docu--mentation/hardware/camera/*). The dimensions of input video from the camera are set at 640×480. The camera is directed to a screen where the testing video is being played. The result shows no difference between using a real camera and using datasets from microSD card.

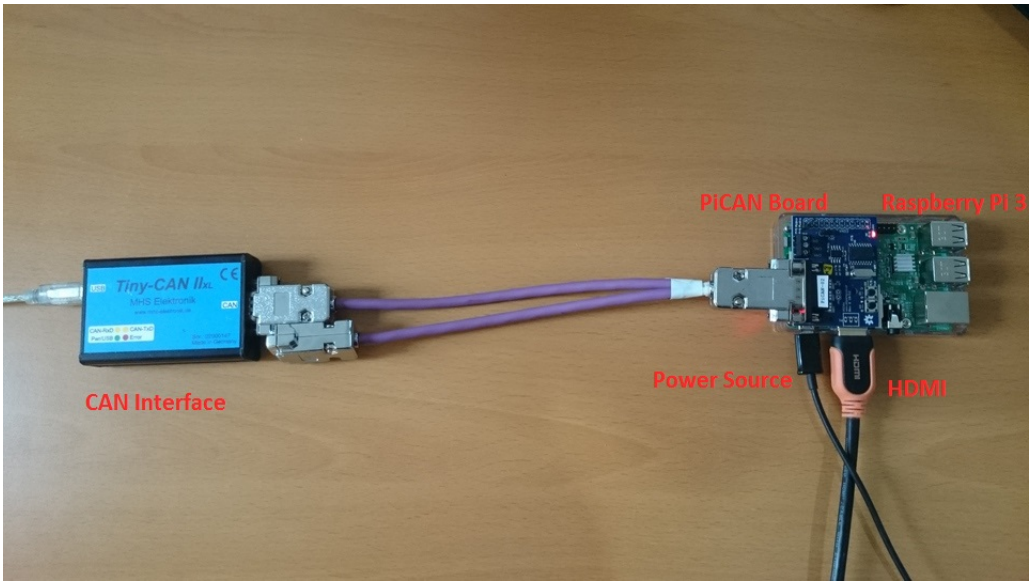


Figure 6.2: Setup of the system for testing and evaluation

6.2 Evaluation criteria

Lane-position detection has been studied for a few decades and results are quite extensive. However, there is a big gap in this research field which is the lack of an effective way to evaluate and compare different researches due to the variation in used datasets. Unlike other research topics in computer vision, such as object or human recognition which use almost identical datasets to test and learn, each lane-position detection algorithm mostly comes with its own datasets. Therefore, the comparison among the studies becomes much more difficult. Some studies have mentioned this problem and tried to propose their solutions such as in the research of McCall and Trivedi [22] and Kim [17].

In this thesis, we try to use criteria and datasets that are able to evaluate the three algorithms based on the initial requirements of the thesis. Besides, these criteria can help in comparing our implemented algorithms with other available algorithms. The criteria include:

1. **Processing speed**: This is the criterion mentioned by all researches. It evaluates the ability to process data and the running speed of the system. This criterion is expressed by two parameters. The first one is the dimensions of input image frames. The second one is the amount of data that the system can process in an unit of time Frames Per Second (FPS). Besides, the processing speed depends heavily on the hardware configuration of the system, therefore this information also needs to be mentioned in the final testing result.

2. **Accuracy**: Two tests are executed for this criterion. First, comparing the output lane marking positions from each algorithm in the same image frame. Second, testing the lane width outputs from each algorithm in a piece of road which has a constant width.

3. **Special cases**: This is a very popular and "open" criterion and used in most researches. It shows the ability of the algorithm to handle challenging scenarios. In the thesis, the workings of three algorithms are compared in different specific situations such as when road surface has many kinds of markings or when lane markings are faded or covered, etc.

6.3 Results

This section presents the test results and evaluation with regard to each criterion.

6.3.1 Processing Speed

As mentioned earlier, processing speed is affected heavily by the system hardware configuration. In this thesis, software is implemented in Raspberry Pi 3 Model B, which is the third generation of the famous single-board computer Raspberry Pi, running the operating system Raspbian. The Raspberry Pi 3 comes with a *1.2GHz 64-bit quad-core ARMv8 CPU* and *1GB of RAM*. Thesis algorithms are developed using C++ programming language and OpenCV library.

Table 6.2 provides the reference processing speeds of some researches for comparison.

Research	Frame Dimensions	Processing Speed	Configuration
Bertozzi and Broggi [5]	512 × 256	10fps for both lane and obstacle detection	the parallel processor for image checking and analysis (PAPRICA) system
Wang et al. [30]	240 × 256	4s for detection and 2fps in tracking	Intel Pentium 3 computer with 128MB of RAM
Kim [17]	176 × 120	10fps	Intel Pentium 4 3GHz computer; algorithm implemented in C++ using OpenCV
Borkar et al. [7]	640 × 480	0.8s/frame	a custom-built Intel-based computer; algorithm implemented in Matlab
Aly [1]	640 × 480	50fps	Intel Core2 2.4GHz computer; algorithm implemented in C++ using OpenCV

Table 6.2: Comparison of the processing speed of some researches

The algorithms in the thesis are tested with the 640 × 360 input images, the dimensions of ROI are 360 × 95. The testing results of processing speed are given in Table 6.3.

From the results shown in Table 6.3, the processing speeds of our implemented algorithms in Raspberry Pi 3 are comparable to other algorithms, even the slowest one - Spline-based Algorithm - can reach around *19 FPS*. This result is important in showing the ability to work totally in real-time of a lane-position detection system implemented in hardware with equivalent configuration.

Algorithm	Processing Speed
Line-based Algorithm	25fps
Spline-based Algorithm	19fps
Hough-based Algorithm	27fps

Table 6.3: Testing results in processing speed of three algorithms in the thesis

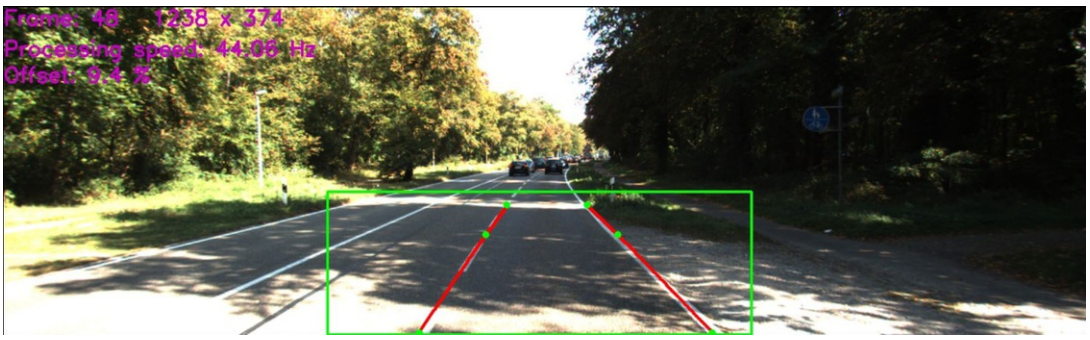
6.3.2 Accuracy

For the accuracy criterion, lane positions output from each algorithm are compared to each other. Because of the lack of ground truth data for each dataset, the result of Line-based Algorithm, which is the best one under observation, is used as the standard, with which the results of Spline-based and Hough-based Algorithm are compared for evaluation. The metrics that are chosen to quantify the errors of Spline-based and Hough-based to Line-based Algorithm are mean absolute error in position and standard deviation of error in position. In each dataset, some parameters such as ROI dimensions are adjusted to obtain the best results.

KITTI Dataset

Figure 6.3 shows the working of the three algorithms in KITTI dataset. The green rectangle is the ROI with dimensions 506 × 164. Two red lines are detected lane markings, the white line is the center of detected lane.

(a) Line-based and Hough-based Lane Detection Algorithm



(b) Spline-based Lane Detection Algorithm

Figure 6.3: Working of the algorithms in KITTI dataset

The graph in Figure 6.4 presents the lane-marking positions detected by three algorithms in each image frame of KITTI dataset. The lower group is the results of left side lane marking, and the upper one is for right side lane marking.

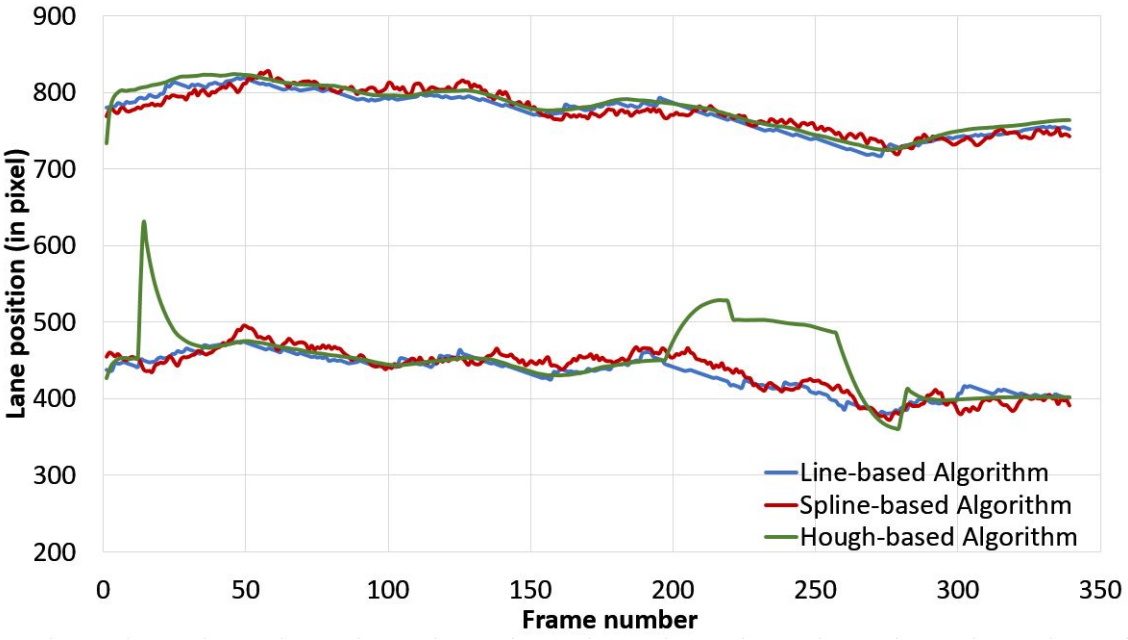


Figure 6.4: Lane position result in KITTI dataset

Table 6.4 describes the quantified comparing result in lane-marking position of Spline-based and Hough-based to Line-based Algorithm. Here, the lane-marking positions output from Line-based Algorithm is considered as ground truth data, and the errors of the output from Spline-based and Hough-based Algorithm are calculated with respect to it.

Algorithm	Mean Absolute Error (cm)	Standard Deviation of Error (cm)
Spline-based Algorithm	3.161	11.395
Hough-based Algorithm	12.924	26.931

Table 6.4: Comparing result of Spline-based and Hough-based Algorithm to Line-based Algorithm - KITTI dataset

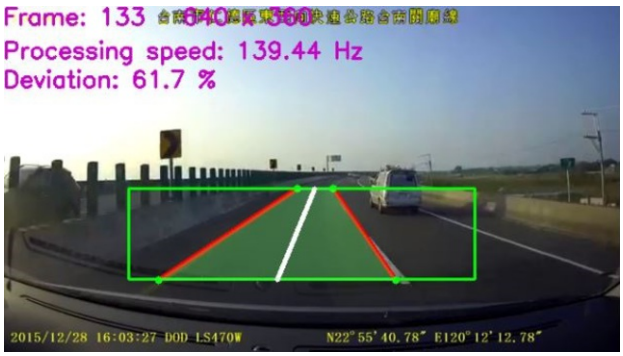
As we can see in the Figure 6.4, Hough-based Algorithm shows the worst result in the case of KITTI dataset, especially it fails from frame 200 to frame 280 (see Figure 6.5). This result can be explained by the susceptible to noises from the environment such as tree shadows, undesired marks on the road surface, of Kalman Filter. While Line-based and Spline-based Algorithm, both use Particle Filter for lane tracking, present quite closed results. The graph of the former is smoother because its lane-marking model - straight line - is simpler than cubic spline model of the latter.

(a) Line-based Lane Detection Algorithm **(b)** Hough-based Lane Detection Algorithm

Figure 6.5: Comparing the working of Line-based and Hough-based Lane Detection Algorithm in frame 235 of KITTI dataset

HwTaiwan Dataset

Figure 6.6 shows the working of the three algorithms in HwTaiwan dataset. The ROI is chosen with dimensions 360×95.



(a) Line-based and Hough-based Lane Detection Algorithm

(b) Spline-based Lane Detection Algorithm

Figure 6.6: Working of the algorithms in HwTaiwan dataset

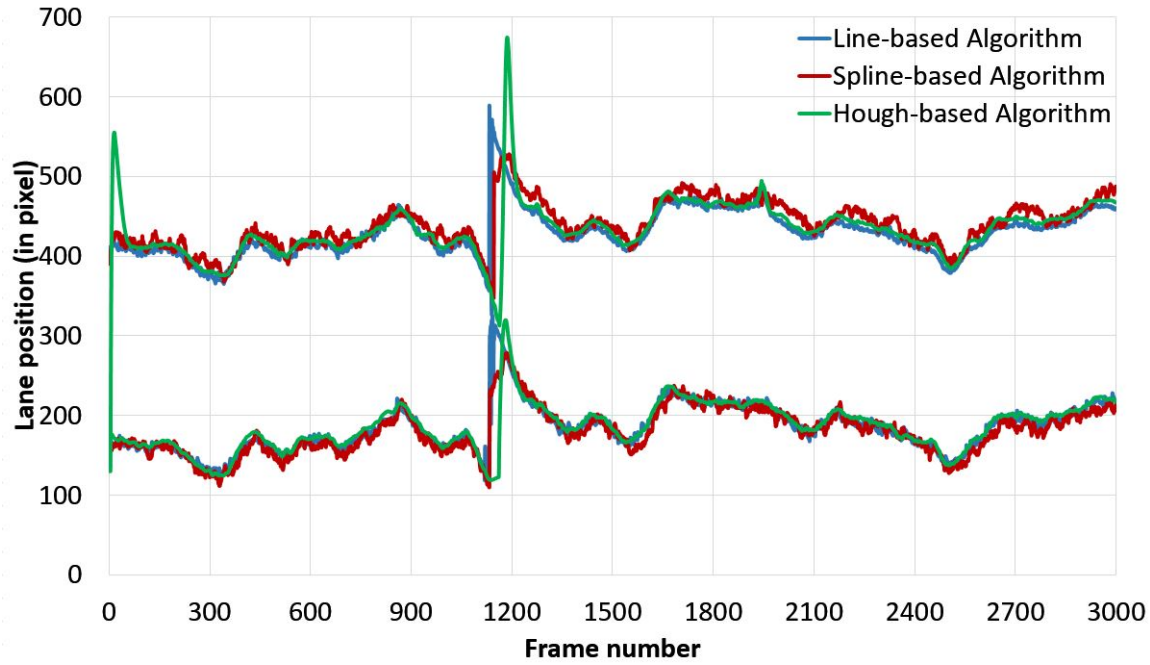


Figure 6.7: Lane position result in HwTaiwan dataset

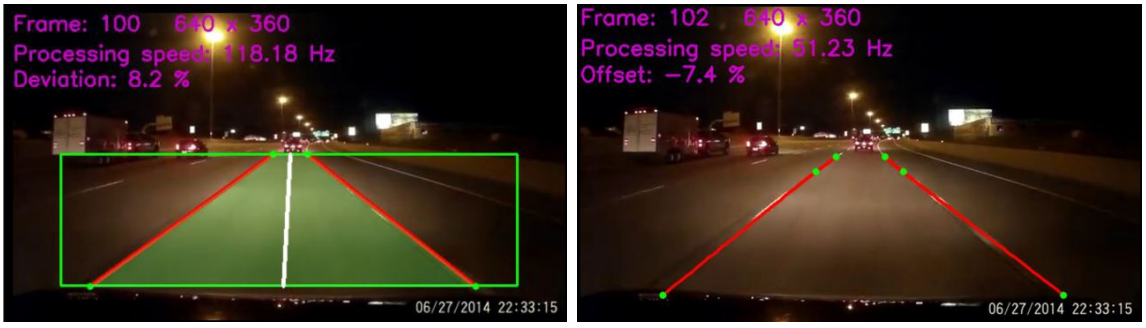
The lane-marking position results of three algorithms in HwTaiwan dataset are compared together in Figure 6.7. HwTaiwan dataset can be considered as an "easy" one, so that the results from all algorithms are quite closed together. We can easily realize the advantage of Kalman Filter in Hough-based Algorithm that is the smoothness of its output in compare to Particle Filter, but when the vehicle changes lane, Kalman Filter requires more time for measurement updating, before it can continue to produce good result. Table 6.5 presents the quantified comparing result of Spline-based and Hough-based to Line-based Algorithm in HwTaiwan dataset.

Algorithm	**Mean Absolute Error (cm)**	**Standard Deviation of Error (cm)**
Spline-based Algorithm	4.387	21.567
Hough-based Algorithm	4.141	35.431

Table 6.5: Comparing result of Spline-based and Hough-based to Line-based Lane Detection Algorithm - HwTaiwan dataset

HwNight Dataset

Figure 6.8 shows the working of the three algorithms in HwNight dataset. The ROI is chosen with dimensions 530×155.



(a) Line-based and Hough-based Algorithm **(b)** Spline-based Algorithm

Figure 6.8: Working of the algorithms in HwNight dataset

The most notable characteristic of HwNight dataset is that, at night, the contrast between lane markings and road surface is better due to their natures of reflection. Besides, there is the lack of unwanted noises caused by shadows of trees or other vehicles. However, the fast changing light intensity is the biggest problem in the

dataset. Light with different colors comes from too many sources such as street lamps, other vehicles, etc. Figure 6.9 presents the output lane-marking positions of the three algorithm.

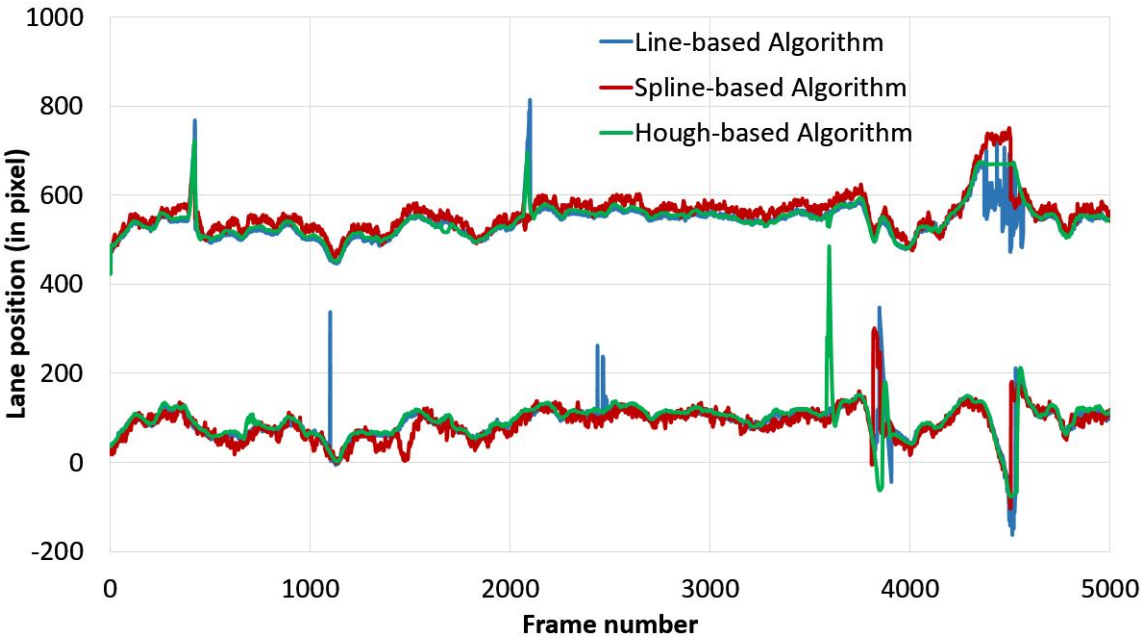


Figure 6.9: Lane position result in HwNight dataset

The quantified comparing result is shown in Table 6.6. Hough-based Algorithm works quite well in this dataset in compare to Line-based and Spline-based Algorithm due to the good extraction of lane-marking evidences at night and road with complicated lane branches. The result of Spline-based Algorithm shows the weakness of IPM technique in the case road has many bumps, this is because of the fact that bumping effect is amplified when image is converted to bird's eye view.

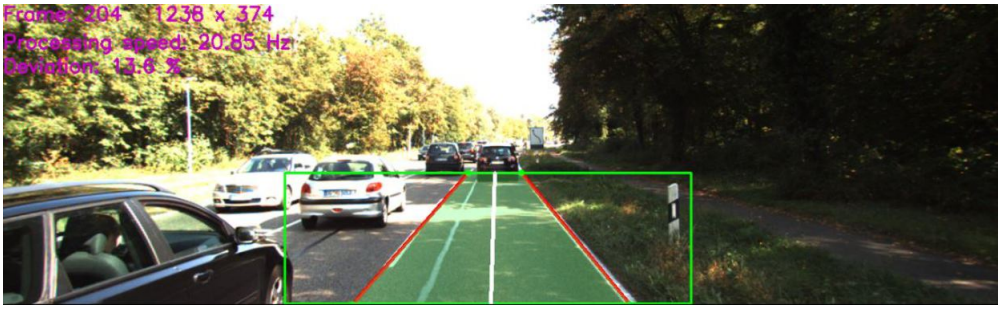
Algorithm	**Mean Absolute Error (cm)**	**Standard Deviation of Error (cm)**
Spline-based Algorithm	5.910	27.502
Hough-based Algorithm	3.725	22.295

Table 6.6: Comparing result of Spline-based and Hough-based to Line-based Lane Detection Algorithm - HwNight dataset

6.3.3 Special Cases

The thesis three algorithms are tested in some special cases and the results are presented as shown in Figure 6.10. These cases include:

- Road surface has unwanted mark as shown in Figure 6.10a, which may be easily confused with lane marking, is handled well by Particle Filter in Line-based and Spline-based Algorithm. Hough-based Algorithm fails in this case.
- Figure 6.10b shows the car stops by traffic jam. All three algorithms still work well and output correct positions of the lane markings.



(a)

(b)

(c)

(d)

Figure 6.10: The results in some special cases

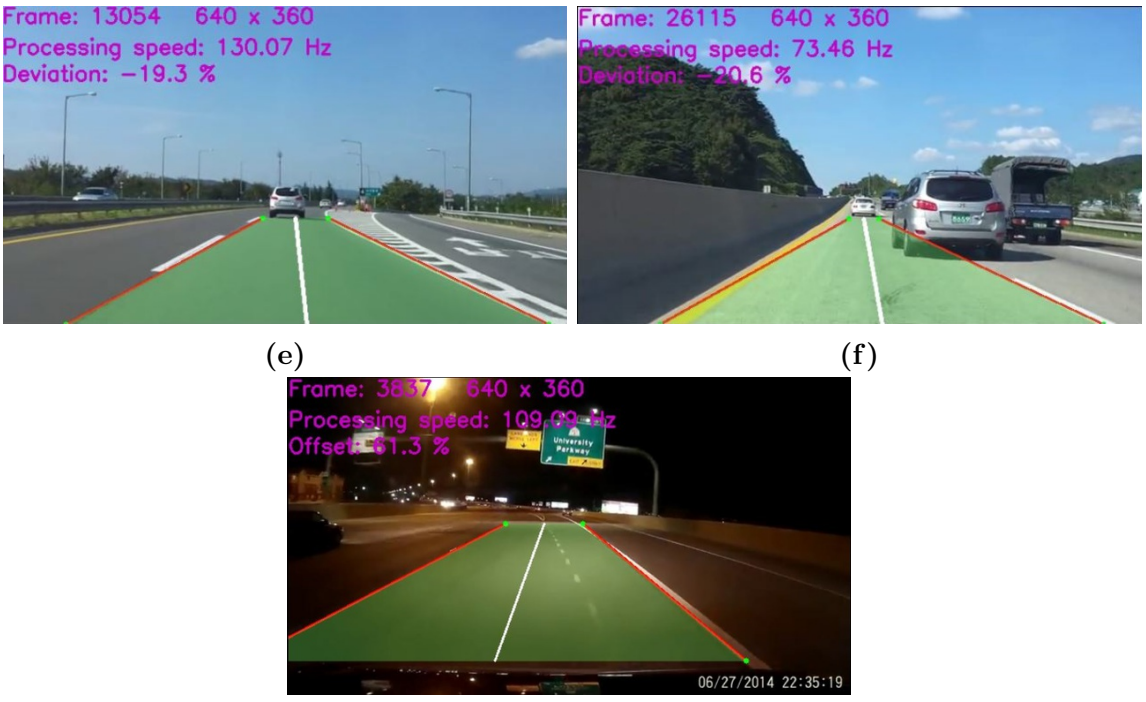


Figure 6.10: The results in some special cases (cont.)

- Road markings with curvature as shown in Figure 6.10c and 6.10d can be represented fine by both straight line and cubic spline model.
- All algorithms are able to handle well the situations that lane marking is complicated as shown in Figure 6.10e or lane marking is covered by other vehicle in Figure 6.10f.
- Hough-based Algorithm performs better in the situation when the car is moving to branch road (Figure 6.10g).

Limitations

In the process of testing, we learn several common limitations of three algorithms in this thesis:

- The lack of an effective method for initialization which is able to handle well all situations when the tracking is failed and the re-detection is triggered, especially for Line-based and Spline-based Algorithm.
- The lack of the precise vehicle motion model and lane width in calculation of all three algorithms. These problems cause the failure of the algorithms when the evidences of lane markings are totally lost.

Some cases of failed detection are shown in Figure 6.11. Most of failed cases are caused by the false evidences from objects on the road such as other vehicles, lane barriers, etc. like in Figure 6.11a, 6.11c and 6.11d. In Figure 6.11b, the solid lane marking of branching lane creates stronger evidences than dash-line lane marking of the main lane. This problem can be solved by detecting multiple lanes at the same time instead of detecting only two lane markings in ROI which create strongest evidences. In Figure 6.11e, the detection of left side lane marking fails because the critical shadow condition makes the evidences of lane marking almost disappear after pre-processing stage.

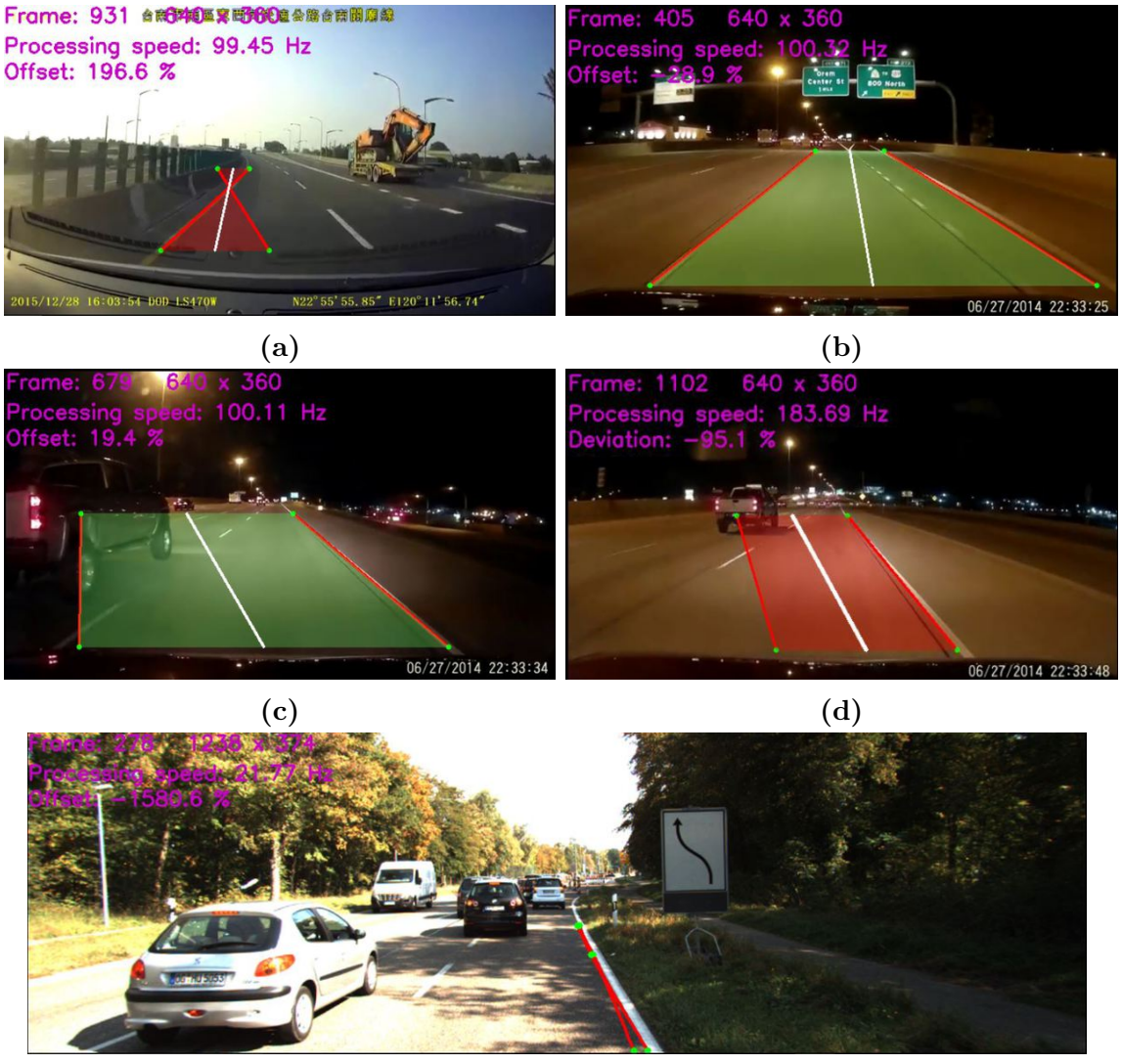


Figure 6.11: Few cases where the algorithms in the thesis failed

6.4 Conclusion

The tests and the received results as presented in this chapter show the ability of our implemented algorithms to meet the initial requirements of the thesis. The lane-position detection system implemented in Raspberry Pi 3 performs well with a good accuracy and a speed enough for real-time working.

During testing process, the advantages and disadvantages of each algorithm are analyzed and compared to each other. Line-based Algorithm is the optimal algorithm in all three because of its good speed and the most accurate result. Meanwhile, Spline-based Algorithm, which uses cubic spline model, shows the best ability of representing lane marking. Besides, it is extremely promising method to focus on in the future development. The workings of both Line-based and Spline-based Algorithm prove the advantage of Particle Filter in lane tracking when they work well even in noisy binary images. Hough-based Algorithm has the highest working speed and a smooth result thanks to Hough Transform and Kalman Filter, however it is the most susceptible to noise. Apparently it performs better than the other algorithms in ideal working condition.

Although the priority of algorithms in the thesis is the speed, which should be good enough to run in real-time in a low-power embedded platform, while the working scenarios are common, in fact, all algorithms under testing show their ability to handle several quite complicated situations such as road with many unwanted marks, lane marking is covered by other vehicles.

7 Conclusion and Future Work

In this thesis, three main tasks have been accomplished:

- Evaluating available state-of-the-art researches in lane-position detection.
- Constructing three different lane-position detection algorithms based on the above evaluation and the initial requirements of the thesis.
- Developing C++ code for the three algorithms and running them in the available hardware.

In the first task, we did extensive research on the current available approaches of lane-position detection problem, focusing on the methods, which use input data from a monocular vision camera. Besides sensing modalities, system objectives and working environment are also important initial considerations in lane detection. Based on the general flowchart proposed by McCall and Trivedi [22], each approach is examined regarding four main points: road modelling, lane feature extraction, post-processing or lane detecting and lane tracking. In each point, we have presented several different notable methods, comparing and discussing their advantages and disadvantages. In fact, the best choice in each point heavily depends on the objectives of the system and the intended working environment. For example, regarding road modeling, a complex road model should not be used in a lane-position system, which is designed to work on highways with moderately simple structure.

In our case, the initial requirement of the thesis is estimating the possibility of implementing a lane-position detection algorithm in our low-power embedded hardware. The intended working environment is rather common, without challenging scenarios. Therefore, the working speed of the algorithm is the top priority. It leads to our choice of used methods. Road modelling is tested with the choice between straight line and cubic spline. A tracking stage is necessary to improve the speed and accuracy of the system. Here, we test both Kalman Filter and Particle Filter. In our all three algorithms, instead of using common edge-based methods, a fast and effective filter is applied for lane feature extraction. The filter is less vulnerable to noises and is highly responsive to horizontal intensity bumps.

In Line-based Algorithm, straight line is chosen to model lane markings. After lane features have been extracted, lane detection is realized by distributing thousands of lane marking hypotheses over the pre-processed image. Each hypothesis is then

assigned a weight using an error function to describe how good it is in representing lane marking. A Particle Filter is utilized for lane tracking. 50 best hypotheses each side from previous step are kept as input for Particle Filter. In each iteration, Particle Filter generates new hypotheses by moving start points and end points based on random numbers from a specific normal distribution. The hypothesis with highest weight will be chosen as the representation of lane marking.

Spline-based Algorithm is the most complicated one in all three algorithms implemented in this thesis. Its realizing steps are relatively similar to Algorithm 1 with the same procedure in lane detecting and lane tracking, which also applies Particle Filter. The difference is that lane marking is modeled by using cubic spline with four control points instead of straight line. Four control points are chosen as following: the first point is in the first row of ROI image, while the fourth point is in the last row, the second and third point are chosen so that the distances between the control points as equal as possible. Additionally, instead of perspective image, the whole process of Spline-based Algorithm is realized in IPM image (or birds-eye-view image), which is generated by using IPM transformation. The benefit of using IPM image is that it removes the perspective effect in the original image, hence, the problems with high curvature and different dimensions of lane markings, together with changing lane width are solved. The IPM transformation is realized through a homography matrix, which is calculated only once and then can be applied to all the other image frames. Therefore, the accuracy in the calculation of homography matrix will greatly affect performance of Spline-based Algorithm.

In all three algorithms of the thesis, Hough-based Algorithm is a very different approach with the highest priority being working speed. To achieve high working speed, Hough Transform is applied to detect lines in the pre-processed image. The best line will be chosen as measurement for lane tracking stage, which is realized by using Kalman Filter. Hough-based Algorithm works effectively in common scenarios with fewer noises such as highways, but fails more in complex situations, in which many unwanted noises are still left after pre-processing stage.

The algorithms implemented in the thesis are tested with three different datasets, which are captured in both highways and urban streets in day and night time. There are three criteria used to evaluate each algorithm, including processing speed, accuracy and special cases. Based on these criteria, the algorithms are compared to each other and evaluated to show their advantages and disadvantages, which algorithm matches the requirements of the thesis and which algorithm should be focused on for further development in the future.

The application of Particle Filter in Line-based and Spline-based Algorithm, instead of Kalman Filter, brings the flexibility and greatly increases the accuracy of

the system. This method is a promising direction for future development. Especially, by being integrated with the information of vehicle motion model, lane width, path planning and GPS, it can handle many challenging situations such as missing lane markings, changes in direction of the vehicle, sudden changes in lane width, crossroads etc. and be able to be applied to Automated Vehicle Control System in reality without significantly increasing in calculation.

The main limitations of the algorithms implemented in the thesis are pointed out as following:

- There is no precise information of vehicle motion model integrated into the calculation.
- The lack of information about lane width in the calculation can make the system failed in cases where lane width changes significantly.
- The lack of an effective method for initializing the system, especially in cases where lane markings are missing for a long period of time, which may lead to an inaccurate result.
- The pre-processing stage in the thesis is fast and effective in most cases, but in challenging situations such as extremely critical light conditions, bad weather, it may be not able to extract the information about lane marking well, and that may lead to the incorrect results of subsequent stages.

Some directions for future development are listed as below:

- Integrating the information about lane width and vehicle motion model into the calculation to increase the performance of the system.
- Developing an effective method for initializing the system
- Extending the lane feature extraction method to handle more challenging scenarios.
- Increasing working speed of the system by applying parallel programming, especially in Spline-based Algorithm.
- Integrating the information of GPS, path planning, etc. in the calculation of the algorithms, so that they can be applied in Automated Vehicle Control System in the further future.

Bibliography

[1] Mohamed Aly. Real time detection of lane markers in urban streets. *IEEE Intelligent Vehicles Symposium*, pages 7–12, June 2008.

[2] N. Apostoloff and A. Zelinsky. Robust vision based lane tracking using multiple cues and particle filtering. *IEEE IV2003 Intelligent Vehicles Symposium*, June 2003.

[3] D.H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111 – 122, 1981.

[4] R. Berriel, E. de Aguiar, V. V. de Souza Filho, and T. Oliveira-Santos. A Particle Filter-based lane marker tracking approach using a cubic spline model. *Proceedings of the 28th SIBGRAPI Conference on Graphics, Patterns and Images*, pages 149–156, 2015.

[5] M. Bertozzi and A. Broggi. GOLD: a parallel real-time stereo vision system for generic obstacle and lane detection. *IEEE Transactions on Image Processing*, 7(1):62–81, January 1998.

[6] A. Borkar, M. Hayes, M. T. Smith, and S. Pankanti. A layered approach to robust lane detection at night. *IEEE Workshop on Computational Intelligence in Vehicles and Vehicular Systems*, pages 51–57, March 2009.

[7] A. Borkar, M. Hayes, and Mark T. Smith. Robust lane detection and tracking with RANSAC and Kalman Filter. *Proceedings of the 16th IEEE International Conference on Image Processing*, pages 3225–3228, 2009.

[8] Gary Bradski and Adrian Kaehler. *Learning OpenCV*. O'Reilly, 2008.

[9] Radu Danescu and Sergiu Nedevschi. Probabilistic lane tracking in difficult road scenarios using stereovision. *IEEE Transactions on Intelligent Transportation Systems*, 10(2):272–282, June 2009.

[10] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, January 1972.

[11] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI Dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[12] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Prentice Hall, third edition, 2008.

[13] N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEE Proceedings F - Radar and Signal Processing*, 140(2):107–113, April 1993.

[14] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[15] H. Hattori. Stereo for 2d visual navigation. *Proceedings of the IEEE Intelligent Vehicles Symposium 2000 (Cat. No.00TH8511)*, pages 31–38, 2000.

[16] Paul V. C. Hough. Method and means for recognizing complex patterns. December 18 1962. URL `https://www.google.com/patents/US3069654`. US Patent 3,069,654.

[17] ZuWhan Kim. Robust lane detection and tracking in challenging scenarios. *IEEE Transactions on Intelligent Transportation Systems*, 9(1):16–26, March 2008.

[18] J. G. Kuk, J. H. An, H. Ki, and N. I. Cho. Fast lane detection & tracking based on hough transform with reduced memory requirement. *13th International IEEE Conference on Intelligent Transportation Systems*, pages 1344–1349, September 2010.

[19] John Leonard, Jonathan How, Seth Teller, Mitch Berger, Stefan Campbell, Gaston Fiore, Luke Fletcher, Emilio Frazzoli, Albert Huang, Sertac Karaman, Olivier Koch, Yoshiaki Kuwata, David Moore, Edwin Olson, Steve Peters, Justin Teo, Robert Truax, Matthew Walter, David Barrett, Alexander Epstein, Keoni Maheloni, Katy Moyer, Troy Jones, Ryan Buckley, Matthew Antone, Robert Galejs, Siddhartha Krishnamurthy, and Jonathan Williams. A perception-driven autonomous urban vehicle. *Journal of Field Robotics*, 25(10): 727–774, October 2008.

[20] P. Lindner, S. Blokzyl, G. Wanielik, and U. Scheunert. Applying multi level processing for robust geometric lane feature extraction. pages 248–254, September 2010.

[21] Peter S. Maybeck. *Stochastic Models, Estimation and Control, Volume 1*, volume 141 of *Mathematics in Science and Engineering*. Academic Press, 1979.

[22] Joel C. McCall and Mohan M. Trivedi. Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation. *IEEE Transactions on Intelligent Transportation Systems*, 7(1):20–37, March 2006.

[23] Marcos Nieto, Jon Arróspide Laborda, and Luis Salgado. Road environment modeling using robust perspective analysis and recursive bayesian segmentation. *Machine Vision and Applications*, 22(6):927–945, 2011.

[24] *The OpenCV Tutorials - Release 2.4.13.1.* OpenCV Dev Team, September 2016.

[25] *The OpenCV Reference Manual - Release 2.4.13.2.* OpenCV Dev Team, February 2017.

[26] S. Sehestedt, S. Kodagoda, A. Alempijevic, and G. Dissanayake. Robust lane detection in urban environments. pages 123–128, October 2007.

[27] *Unfallentwicklung auf deutschen Straßen 2015.* Statistisches Bundesamt, Wiesbaden, June 2016.

[28] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics.* Intelligent Robotics and Autonomous Agents. 2005.

[29] Vincent Voisin, Manuel Avila, Bruno Emile, Stephane Begot, and Jean-Christophe Bardet. Road markings detection and tracking using Hough Transform and Kalman Filter. *Proceedings of the 7th International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 76–83, 2005.

[30] Yue Wang, Eam Khwang Teoh, and Dinggang Shen. Lane detection and tracking using B-Snake. *Image and Vision Computing*, 22(4):269 – 280, 2004.

[31] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. Technical report, Chapel Hill, NC, USA, 1995.

[32] B. F. Wu, C. T. Lin, and Y. L. Chen. Dynamic calibration and occlusion handling algorithms for lane tracking. *IEEE Transactions on Industrial Electronics*, 56(5):1757–1773, May 2009.

TECHNISCHE UNIVERSITÄT CHEMNITZ

Studentenservice – Zentrales Prüfungsamt

Selbstständigkeitserklärung

Name: Nguyen Vorname: Trung Bao geb. am: 21.07.1989 Matr.-Nr.: 362252	**Bitte beachten:** 1. Bitte binden Sie dieses Blatt am Ende Ihrer Arbeit ein.

Selbstständigkeitserklärung*

Ich erkläre gegenüber der Technischen Universität Chemnitz, dass ich die vorliegende Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keinem anderen Prüfer als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Datum: 13.06.2017 Unterschrift: .....

---

* Statement of Authorship

I hereby certify to the Technische Universität Chemnitz that this thesis is all my own work and uses no external material other than that acknowledged in the text.

This work contains no plagiarism and all sentences or passages directly quoted from other people's work or including content derived from such work have been specifically credited to the authors and sources.

This paper has neither been submitted in the same or a similar form to any other examiner nor for the award of any other degree, nor has it previously been published.